Verwenden des TKG-Diensts mit der vSphere IaaS-Steuerungsebene

Update 3

VMware vSphere 8.0
VMware vCenter 8.0
VMware ESXi 8.0

vmware
by Broadcom

Die aktuellste technische Dokumentation finden Sie auf der VMware by Broadcom-Website unter:

https://docs.vmware.com/de/

VMware by Broadcom
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Inhalt

Verwenden des TKG-Diensts mit der vSphere IaaS-Steuerungsebene

Diese Dokumentation bietet Ihnen Informationen zur Verwaltung des Lebenszyklus von Tanzu Kubernetes Grid-Clustern auf dem Supervisor in vSphere IaaS control plane.

Mit Tanzu Kubernetes Grid auf Supervisor können Sie verschiedene Arten von Arbeitslastclustern bereitstellen, darunter Tanzu Kubernetes-Cluster mit genau definierten Vorgaben und Cluster Class-Cluster mit umfassenden Definitionsoptionen. Durch die Bereitstellung von Supervisor in vSphere-Zonen können Sie hochverfügbare TKG-Cluster bereitstellen, die fehlertolerante Arbeitslasten unterstützen. Das aktualisierte Tanzu Kubernetes-Versionsformat integriert nativ das Tanzu-Paket-Repository und unterstützt mehrere Betriebssysteme. Sie können mit vCenter Single Sign-On und den Kubernetes-CLI-Tools für vSphere oder über einen externen Identitätsanbieter und die Tanzu-CLI auf TKG-Cluster auf Supervisor zugreifen und diese verwalten.

Zielgruppe

Diese Dokumentation ist für vSphere-Administratoren und Kubernetes-Operatoren bestimmt, die den Lebenszyklus von TKG-Clustern auf dem Supervisor in vSphere IaaS control plane bereitstellen und verwalten möchten.

Anwendbare vSphere- und TKG-Versionen

Sofern nicht anders angegeben, gilt diese Dokumentation für die vSphere 8-Version von vSphere IaaS control plane und wird aktualisiert, um TKG v3.x auf dem Supervisor zu unterstützen. Dokumentationsaktualisierungen sind kumulativ. Die archivierte Dokumentation für frühere Versionen steht auf der VMware-Dokumentationsseite unter der Überschrift **Archivpakete** zum Download zur Verfügung.

Aktualisierte Informationen

1

Diese Dokumentation wird bei Bedarf regelmäßig mit neuen Informationen und Fehlerbehebungen aktualisiert.

Die Tabelle enthält den Updateverlauf für dieser Dokumentation.

Revision	Beschreibung
25. Juni 2024	Updates für die Dokumentation zur Unterstützung von vSphere 8 Update 3 und TKG-Dienst 3.0, einschließlich: ■ Kapitel 3 Installieren und Aktualisieren des TKG-Diensts ■ Kapitel 10 Automatische Skalierung von TKG-Dienstclustern ■ Kapitel 20 Sichern und Wiederherstellen von TKG-Dienstclustern und -Arbeitslasten ■ Aktivieren des Antrea-NSX-Adapters für einen TKG-Dienstcluster ■ Installieren des NSX-Verwaltungs-Proxy-Diensts ■ Kapitel 11 Installieren von Standardpaketen auf TKG-Dienst-Clustern ■ Konfigurieren von MachineHealthCheck für v1beta1-Cluster ■ Cluster-API v1beta1 (Updates) ■ controlPlaneCertificateRotation (aktualisiert) ■ podSecurityStandard (neu) ■ v1beta1-Beispiel: Cluster basierend auf einer benutzerdefinierten ClusterClass (vSphere 8 U2- und höherer Workflow) (Updates) ■ Installieren und Konfigurieren des Velero-Plug-In für vSphere auf einem TKG-Cluster (Updates) ■ Manuelles Skalieren eines Clusters mithilfe von „Kubectl“ (Updates)
8. März 2024	■ Die Anforderungen an CSI-Snapshots wurden mit dem Hinweis aktualisiert, dass vSphere 8.0 Update 2+ mit TKr v1.26 und höher als Mindestversionen verwendet werden. Weitere Informationen hierzu finden Sie unter Kapitel 15 Erstellen von Snapshots in einem TKG-Dienst-Cluster. ■ Anweisungen zum Sichern und Wiederherstellen von TKGS-Clusterarbeitslasten mithilfe von Velero mit CSI-Snapshot wurden hinzugefügt. Weitere Informationen hierzu finden Sie unter Sichern und Wiederherstellen mithilfe von Velero mit CSI-Snapshot. ■ Aktualisieren Sie das Beispiel und die Überprüfungsschritte für die Bereitstellung eines v1beta1-API-Clusters mit einem FQDN. Weitere Informationen hierzu finden Sie unter v1beta1-Beispiel: Cluster mit FQDN. ■ Die Beschreibung und die Kubernetes-Einschränkung der Rollenberechtigung „Lesezugriff“ für vSphere-Namespaces wurden aktualisiert. Weitere Informationen hierzu finden Sie unter Rollenberechtigungen und -bindungen. ■ Explizite Anweisungen zum Installieren des vertrauenswürdigen CA-Root-Zertifikats von vCenter auf einem Linux-Clienthost wurden hinzugefügt. Weitere Informationen hierzu finden Sie unter Herunterladen der vertrauenswürdigen CA-Root-Zertifikate für vCenter und Installieren dieser Zertifikate auf einem Ubuntu-Client.

Revision	Beschreibung
29. FEB 2024	■ Die Dokumentation wurde aktualisiert, um die GA-Version von vSphere 8 U2 P03 zu unterstützen. ■ Eine Dokumentation zur Unterstützung bei der Erstellung eines Clusters mithilfe eines FQDN wurde hinzugefügt. Weitere Informationen finden Sie unter Cluster-API v1beta1 und v1beta1-Beispiel: Cluster mit FQDN. ■ Die Supervisor-Anmeldebefehle wurden mit dem Hinweis aktualisiert, dass der FQDN bei aktivierter Option verwendet werden kann. ■ Der PSA-Dokumentation wurden zusätzliche Beispiele hinzugefügt. Weitere Informationen hierzu finden Sie unter Konfigurieren von PSA für TKR 1.25 und höher. ■ Anweisungen zum Installieren des NVIDIA-Netzwerkoperators auf einem TKGS-Arbeitslastcluster zum Ausführen von vGPU-Arbeitslasten wurden hinzugefügt. Weitere Informationen hierzu finden Sie unter Cluster-Operator-Workflow für die Bereitstellung von KI-/ML-Arbeitslasten in TKG-Dienstclustern. ■ Inhalte zur Verwendung des Assistenten zum Erstellen von VM-Klassen, insbesondere zum Erstellen einer benutzerdefinierten VM-Klasse für NVIDIA GRIP vGPUs, wurden hinzugefügt. Weitere Informationen hierzu finden Sie unter Kapitel 13 Bereitstellen von AI/ML-Arbeitslasten in TKG-Dienst-Clustern. ■ #unique_22 Und Installieren und Konfigurieren des Velero-Plug-In für vSphere auf einem TKG-Cluster wurden mit Links zur Kompatibilitätsmatrix für die Velero-Version aktualisiert.
2. Februar 2024	■ Die Dokumentation zur Bereitstellung von TKG-Clustern wurde mit dem Hinweis aktualisiert, dass die v1alpha2-API auf vSphere 8 automatisch in die v1alpha3-API umgewandelt wird. Weitere Informationen hierzu finden Sie unter Über die TKG-Clusterbereitstellung.
31. Januar 2024	■ Verschiedene Fehlerkorrekturen und Bearbeitungen.
19. Januar 2024	■ Das Thema zum Integrieren eines TKGS-Clusters in eine private Containerregistrierung wurde aktualisiert. Weitere Informationen hierzu finden Sie unter Integrieren von TKG-Dienst-Clustern in eine private Containerregistrierung.
18. Januar 2024	■ Cluster-API v1beta1 wurde aktualisiert, um sicherzustellen, dass die `defaultRegistrySecret`-Variable für die Verwendung mit der eingebetteten Harbor-Registrierung reserviert ist.
16. Januar 2024	■ Kapitel 15 Erstellen von Snapshots in einem TKG-Dienst-Cluster wurde mit dem Hinweis aktualisiert , dass die Repository-Version v2023.9.19 oder höher des TKG-Standardpakets verwendet werden muss.
8. Januar 2024	■ Verschiedene kleinere Updates.
2. Januar 2024	■ Verschiedene kleinere Updates.
20. Dezember 2023	■ Verwenden von Kubernetes-Versionen mit TKG-Dienstclustern wurde mit dem Link zur Dokumentation für die Erstellung benutzerdefinierter TKr-Images aktualisiert. ■ Verschiedene kleinere Updates.
6. Dezember 2023	■ Die Versionshinweise zu TKR wurden mit neuen Tanzu Kubernetes Releases aktualisiert. ■ Konfigurieren von PSA für TKR 1.25 und höher wurde mit dem Hinweis aktualisiert, dass PSA in System-Namespaces nicht geändert werden kann. ■ Grundlegendes zum Modell für parallele Updates für TKG-Dienstcluster wurde mit Überlegungen zur Verwendung mehrerer Knotenpools aktualisiert.

Revision	Beschreibung
21. NOV 2023	■ v1alpha3-Beispiel: TKC mit routingfähigem Pod-Netzwerk und v1beta1-Beispiel: Cluster mit routingfähigem Pods-Netzwerk wurden mit dem Hinweis aktualisiert, dass /21 die Mindestgröße für ein routingfähiges Pod-Netzwerk ist. ■ Grundlegendes zum Modell für parallele Updates für TKG-Dienstcluster wurde mit dem geeigneten Zeitpunkt zum Auslösen eines parallelen Updates für vSphere 7 Supervisor-Updates aktualisiert. ■ Fehlerbehebung bei der Container-Bereitstellung wurde aktualisiert, um anzugeben, wann PSP für TKr benötigt wird.
13. November 2023	■ Die Dokumentation wurde mit dem Hinweis aktualisiert, dass ein gültiger Konfigurationsschlüssel aus alphanumerischen Zeichen, Bindestrichen, Unterstrichen oder Punkten bestehen muss (wie z. B. „key.name“, „KEY_NAME“ oder „key-name“).
7. November 2023	■ Das Thema zum Upgrade der TKr-Versionszeichenfolge wurde aktualisiert. Weitere Informationen hierzu finden Sie unter Aktualisieren eines TKG-Clusters durch Bearbeiten der TKR-Version.
16. Oktober 2023	■ Links wurden repariert.
13. Oktober 2023	■ Die Anweisungen zum Skalieren eines TKG-Clusters wurden aktualisiert und enthalten nun Beispiele für die Verwendung der v1beta1-API. Weitere Informationen hierzu finden Sie unter Manuelles Skalieren eines Clusters mithilfe von „Kubectl“.
11. Oktober 2023	■ Die Anweisungen für die Verbindung zu einem TKG-Cluster als OIDC-Benutzer unter Verwendung der Tanzu-CLI wurden aktualisiert. Weitere Informationen hierzu finden Sie unter Herstellen einer Verbindung zu einem TKG-Cluster als OIDC-Benutzer mit der Tanzu-CLI.
9. Oktober 2023	■ Die Anweisungen zum Verknüpfen der Inhaltsbibliothek mit vSphere-Namespaces wurden aktualisiert. Weitere Informationen hierzu finden Sie unter Verknüpfen der TKR-Inhaltsbibliothek mit dem TKG-Dienst.
3. Oktober 2023	■ Die Dokumentation zur benutzerdefinierten ClusterClass von vSphere 8 U2 wurde aktualisiert und enthält nun Anmerkungen zum Erstellen einer nicht verwalteten benutzerdefinierten ClusterClass. Weitere Informationen hierzu finden Sie unter v1beta1-Beispiel: Cluster basierend auf einer benutzerdefinierten ClusterClass (vSphere 8 U2- und höherer Workflow).
21. September 2023	Die Dokumentation für die Funktionen der Version vSphere 8 U2 wurde aktualisiert, einschließlich: ■ Erstellen von Volume-Snapshots für TKG-Cluster. Weitere Informationen hierzu finden Sie unter Kapitel 15 Erstellen von Snapshots in einem TKG-Dienst-Cluster. ■ Installieren von-Paketen auf TKG-Clustern. Weitere Informationen hierzu finden Sie unter Kapitel 11 Installieren von Standardpaketen auf TKG-Dienst-Clustern. ■ Bereitstellung benutzerdefinierter ClusterClass-Cluster mithilfe der v1beta1-API. Weitere Informationen hierzu finden Sie unter Verwenden der v1beta1-API von Clustern. ■ Überlegungen zum parallelen Update für TKG-Cluster. Weitere Informationen hierzu finden Sie unter Grundlegendes zum Modell für parallele Updates für TKG-Dienstcluster. ■ Verschiedene Wartungsaktualisierungen und -bearbeitungen. **Hinweis** Lesen Sie auch die Versionshinweise für vSphere IaaS control plane und Tanzu Kubernetes-Versionen.

Revision	Beschreibung
16. August 2023	■ Aktualisierung der Beispiele für die Bereitstellung von TKG-Clustern für v1alpha3 TKC mit routingfähigen Pods und v1beta1 Cluster mit benutzerdefinierter ClusterClass. Weitere Informationen hierzu finden Sie unter Kapitel 7 Bereitstellen von TKG-Dienstclustern.
15. August 2023	■ Die Dokumentation zur Installation des Tanzu-Pakets wurde aktualisiert. Weitere Informationen hierzu finden Sie unter Kapitel 11 Installieren von Standardpaketen auf TKG-Dienst-Clustern. ■ Kleinere Änderungen und Korrekturen.
10. August 2023	■ Es wurde eine Dokumentation hinzugefügt, wie man Tanzu-Paket-Images von der öffentlichen Harbor-Registrierung an eine private Harbor-Registrierung weitergibt, die als Supervisor-Dienst ausgeführt wird. Weitere Informationen hierzu finden Sie unter Kapitel 14 Verwenden privater Registrierungen mit TKG-Dienstclustern. ■ Kleinere Änderungen und Korrekturen.
7. August 2023	■ Dokumentation für ExternalTrafficPolicy und LoadBalancerSourceRanges für Diensttyp: LoadBalancer mit NSX entfernt. Diese Funktionen werden nicht unterstützt.
2. August 2023	■ Es wurde ein Thema zum Herstellen einer Verbindung mit TKG-Clustern mithilfe der Tanzu-CLI und kubectl hinzugefügt. Weitere Informationen hierzu finden Sie unter #unique_36. ■ Kleinere Änderungen und Korrekturen.
31. Juli 2023	■ Kleinere Änderungen und Korrekturen.
27. JULI 2023	■ Die Dokumentation wurde aktualisiert, um TKG 2.2 zu unterstützen.
21. JULI 2023	■ Es wurde eine Liste der erforderlichen Berechtigungen für die Verwaltung der Inhaltsbibliothek für Tanzu Kubernetes-Versionen hinzugefügt.
12. JULI 2023	■ Der Name der Editor- und Viewer-Rollen, die bearbeitet und angezeigt werden sollen, wurde aktualisiert. Weitere Informationen finden Sie unter Informationen zur Identitäts- und Zugriffsverwaltung für TKG-Dienst-Cluster.
10. JULI 2023	■ Es wurden Anleitungen zum Bearbeiten der Inhaltsbibliothek hinzugefügt, die von TKG-Clustern auf Supervisor verwendet werden kann. Weitere Informationen finden Sie unter Bearbeiten einer vorhandenen Inhaltsbibliothek. ■ Das Thema zum Erfassen von Support-Paketen für TKG-Cluster auf Supervisor wurde aktualisiert. Weitere Informationen finden Sie unter Abrufen von Protokollen zur Fehlerbehebung bei TKG-Clustern auf Supervisor.
7. JULI 2023	■ Das Thema wurde aktualisiert, in dem die beiden verschiedenen Clustertypen erläutert werden, die Sie mithilfe von TKG 2.0 auf Supervisor bereitstellen können. Weitere Informationen finden Sie unter Über die TKG-Clusterbereitstellung. ■ Es wurde ein Thema mit Überlegungen zur Bereitstellung von Clustern mit benutzerdefinierten Knoten-Volume-Mounts hinzugefügt. Weitere Informationen finden Sie unter Überlegungen zur Verwendung von Knoten-Volume-Bereitstellungen.
3. JULI 2023	■ Das Thema zum Erstellen einer vSphere IaaS control plane-Operatorengruppe und Rolle wurde aktualisiert. Weitere Informationen finden Sie unter Erstellen einer dedizierten Gruppe und Rolle für Plattformoperatoren.

Revision	Beschreibung
30. JUNI 2023	■ Das Skript für die Installation von Cert Manager wurde aktualisiert. Weitere Informationen finden Sie unter #unique_41. ■ Es wurden Anweisungen zum Registrieren einer Instanz von Tanzu Mission Control Self-Managed bei Supervisor hinzugefügt. Weitere Informationen finden Sie unter Registrieren von Tanzu Mission Control Self-Managed bei Supervisor. ■ Die VM-Jump-Host-Anweisung für vDS-Umgebungen wurde aktualisiert. Weitere Informationen finden Sie unter Erstellen einer VM für einen Linux-Jump-Host.
26. JUNI 2023	■ Die Liste der TKG-Cluster-APIs wurde mit Details zu aktualisiert, die von vSphere 8 Supervisor unterstützt werden. Weitere Informationen finden Sie unter Über die TKG-Clusterbereitstellung. ■ Die Berechtigungsliste zum Erstellen einer dedizierten Rolle „TKG-Cluster-Operatoren“ wurde aktualisiert. Weitere Informationen finden Sie unter Erstellen einer dedizierten Gruppe und Rolle für Plattformoperatoren. ■ Der Befehl zum Installieren von Contour mithilfe der Tanzu-CLI wurden aktualisiert. Weitere Informationen finden Sie unter Erstellen des Paket-Repositorys.
13. JUNI 2023	■ Es wurde ein Thema zum Erstellen einer dedizierten TKG-Cluster-Operatorengruppe und -Rolle hinzugefügt. Weitere Informationen finden Sie unter Erstellen einer dedizierten Gruppe und Rolle für Plattformoperatoren. ■ Das Thema zum Erstellen eines v1beta1-Clusters mithilfe eines routingfähigen Pod-Netzwerks wurde aktualisiert. Weitere Informationen finden Sie unter v1beta1-Beispiel: Cluster mit routingfähigem Pods-Netzwerk. ■ Das Thema zum Erstellen eines v1alpha3-Tanzu Kubernetes-Clusters mit einem routingfähigen Pod-Netzwerk wurde aufgrund eines bekannten Problems entfernt. Weitere Details finden Sie in den Versionshinweisen.
9. JUNI 2023	■ Es wurde ein Thema zum Entfernen eines vSphere-Namespace aus Supervisor hinzugefügt, das voraussieht, dass alle in diesem vSphere-Namespace bereitgestellten TKG-Cluster gelöscht werden. Weitere Informationen finden Sie unter Entfernen einer vSphere-Namespace.
6. JUNI 2023	■ Die vSphere IaaS control plane-Dokumentationszuordnungsgrafik zum Navigieren in den verschiedenen Publikationen, aus denen sich der Dokumentsatz zusammensetzt, wurde hinzugefügt. Weitere Informationen finden Sie unter Verwenden des TKG-Diensts mit der vSphere IaaS-Steuerungsebene.
5. JUNI 2023	■ Redaktionelle Änderungen an verschiedenen Themen in den folgenden Abschnitten vorgenommen: Kapitel 2 Ausführen von TKG-Dienstclustern und Kapitel 9 Aktualisieren von TKG-Dienstclustern. ■ Der Verwalten von TLS-Zertifikaten für TKG-Dienstcluster wurde mit Verweisen auf vSphere with Tanzu Certificate Guide und andere Änderungen aktualisiert.
1. JUNI 2023	■ Aktualisierte Beispiele für die Clusterbereitstellung. Weitere Informationen finden Sie unter Verwenden der v1alpha3-API von TanzuKubernetesCluster und Verwenden der v1beta1-API von Clustern. ■ Aktualisierte Beispiele für die Clusterskalierung. Weitere Informationen finden Sie unter Manuelles Skalieren eines Clusters mithilfe von „Kubectl“.
26. MAI 2023	■ Das Thema „Paralleles Update für TKG-Cluster“ wurde mit zusätzlichen Details aktualisiert. Weitere Informationen finden Sie unter Grundlegendes zum Modell für parallele Updates für TKG-Dienstcluster.
23. MAI 2023	■ Die Dokumentation wurde umgestaltet, um TKG 2.0 anstelle von TKG 2 zu verwenden.

Revision	Beschreibung
22. MAI 2023	■ Der Befehl zum Erstellen einer Namespace-Pod-Sicherheitsrichtlinie wurde aktualisiert. Weitere Informationen finden Sie unter Anwenden der standardmäßigen Pod-Sicherheitsrichtlinie auf TKG-Dienstcluster. ■ Die Dokumentation des Upgrade-Workflows wurde aktualisiert. Weitere Informationen finden Sie unter #unique_51.
12. MAI 2023	■ Es wurde ein Hinweis hinzugefügt, der besagt, dass Sie neue TKG 2-Cluster nur auf einer Supervisor-Instanz bereitstellen können, die in drei vSphere Zonen bereitgestellt wird.
11. MAI 2023	■ Es wurde ein Thema zur Erweiterung persistenter Volumes hinzugefügt. Weitere Informationen finden Sie unter Erweiterung persistenter Volumes für TKG-Dienst-Cluster.
9. MAI 2023	■ Die Zonentopologie-Grafik wurde aktualisiert, um anzugeben, dass die Pods in diesem Beispiel zonenübergreifend bereitgestellt werden. Weitere Informationen finden Sie unter Referenzarchitekturen für TKG-Dienst-Cluster. ■ Verwenden von vSphere-Namespaces mit TKG-Dienstclustern wurde mit zusätzlichen Speicherdetails aktualisiert. ■ Die Kapitel 8 Betreiben von TKG-Dienstclustern wurde mit zusätzlichen Details zur Überwachung des persistenten Speichers aktualisiert.
4. MAI 2023	■ Die Erstellen des Paket-Repositorys wurde aktualisiert, um `--package` zu verwenden. ■ Konfigurieren von PSP für TKR 1.24 und früher aktualisiert, um klarzustellen, dass PSPs nach wie vor für TKG 2-Cluster erforderlich sind und dass der Zugangscontroller „Pod-Sicherheit“ nicht unterstützt wird. ■ #unique_22 aktualisiert.
1. MAI 2023	■ Link korrigiert.
4. APR 2023	■ Es wurden allgemeine Updates und Fehlerkorrekturen für die Version vSphere 8 Update 1 vorgenommen. ■ Verwenden von Kubernetes-Versionen mit TKG-Dienstclustern wurde für die Version vSphere 8 U1 aktualisiert, einschließlich der Überprüfung der TKR-Kompatibilität mit Supervisor und mit TKG 2, der Entfernung des Suffixes `-zshippable` aus der TKR NAME-Zeichenfolge und der Erstellung Ihrer eigenen TKR. ■ Die Dokumentation der Cluster-API v1beta1 -Spezifikation wurde aktualisiert. ■ Integrieren von TKG-Dienst-Clustern in eine private Containerregistrierung mit Beispielen aktualisiert. ■ Konfigurieren eines Texteditors für Kubectl wurde mit zusätzlichen Beispielen für Windows aktualisiert. ■ In Sichern und Wiederherstellen von TKG-Cluster-Arbeitslasten Supervisor mit eigenständigem Velero und Restic wurden Fehlerkorrekturen durchgeführt.

Ausführen von TKG-Dienstclustern

2

In diesem Abschnitt werden die Komponenten beschrieben und die Anforderungen für die Ausführung von TKG-Dienstclustern aufgelistet.

Lesen Sie als Nächstes die folgenden Themen:

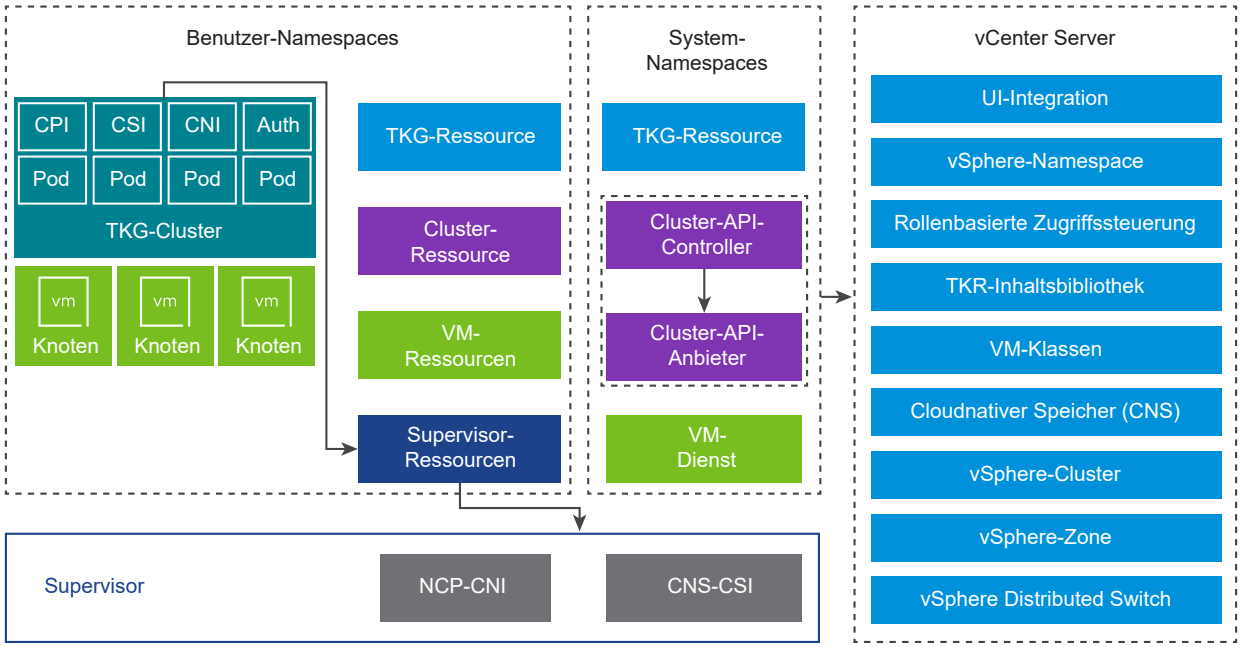
- Komponenten für TKG-Dienst
- Bereitstellen von TKG-Dienst-Clustern
- Referenzarchitekturen für TKG-Dienst-Cluster

Komponenten für TKG-Dienst

TKG-Dienst bietet Self-Service-Lebenszyklusverwaltung von Kubernetes-Arbeitslastclustern. Der TKG-Dienst mit Supervisor ist für die vSphere-Umgebung optimiert und lässt sich in die zugrunde liegende Infrastruktur, einschließlich vCenter, ESXi, virtuelle Netzwerke und Cloud Native Storage, integrieren. Mit dem TKG-Dienst können Sie konforme Kubernetes-Cluster bereitstellen und Upstream-Parallelität beibehalten.

Der TKG-Dienst enthält mehrere Komponenten, die in die vSphere IaaS control plane integriert sind.

Abbildung 2-1. Komponenten für TKG-Dienst



Arbeitslastverwaltung

Die Arbeitslastverwaltung ist eine VMware-Lösung, die eine Kubernetes-Steuerungsebene in der nativen vSphere-Infrastruktur bereitstellt. Durch die Aktivierung der Arbeitslastverwaltung können Sie einen oder mehrere Supervisoren in Ihrer vSphere-Umgebung bereitstellen. Die Arbeitslastverwaltung ist im Lieferumfang von vCenter Server enthalten, aber separat lizenziert.

Supervisor

Supervisor ist ein Kubernetes-Cluster, der als Steuerungsebene für die Verwaltung von Arbeitslastclustern dient. Sie konfigurieren Supervisor zur Unterstützung von Entwicklern, die Tanzu Kubernetes Grid-Cluster bereitstellen und betreiben.

Supervisor-Bereitstellung: vSphere-Cluster

Herkömmlicherweise wird Supervisor ist auf einem einzelnen vSphere-Cluster bereitgestellt. Ein vSphere-Cluster ist eine Sammlung von ESXi-Hosts, die von einem vCenter Server verwaltet werden. Ein vSphere-Cluster muss mit bestimmten Funktionen zur Unterstützung von Supervisor konfiguriert werden:

- Mehrere ESXi Hosts, die über einen vSphere Distributed Switch (VDS) verbunden sind
- Freigegebener Speicher ist konfiguriert, wie vSAN oder NFS
- vSphere HA und vollautomatisierte DRS sind aktiviert
- Lifecycle Manager ist aktiviert
- Netzwerk ist konfiguriert, entweder NSX mit eingebettetem Lastausgleichsdienst oder VDS mit externem Lastausgleichsdienst

Architektonisch sollten Sie für Produktionsbereitstellungen von TKG auf dem Supervisor den Host oder Cluster der Management Plane, auf dem vCenter Server ausgeführt wird, vom Cluster der Rechnerebene trennen, in dem Supervisor aktiviert wird. Wenn Sie einen vSphere-Cluster zum Hosten vCenter Server verwenden, sollte für diesen Cluster DRS nicht aktiviert sein. In der vCenter-Dokumentation finden Sie weitere Details.

Supervisor-Bereitstellung: vSphere-Zonen

In vSphere 8 werden vSphere-Zonen eingeführt. Sie können vSphere-Cluster zu vSphere-Zonen zuweisen, um Hochverfügbarkeit und Fault Tolerance für Supervisor bereitzustellen. Durch die Supervisor-Bereitstellung in verschiedenen vSphere-Zonen können Sie TKG-Cluster in bestimmten Verfügbarkeitszonen bereitstellen.

Bei einer Bereitstellung von Supervisor auf einem einzelnen vSphere-Cluster besteht eine 1:1-Beziehung zwischen dem Supervisor und dem vSphere-Cluster. Bei einer Zonenbereitstellung mit Supervisor erstreckt sich der Supervisor über drei vSphere-Cluster, um Hochverfügbarkeit und Fehlerdomänen für TKG-Cluster auf Supervisor bereitzustellen.

Ein vSphere-Administrator erstellt die drei vSphere-Zonen und ordnet sie einem vSphere-Cluster zu. Für eine Supervisor-Bereitstellung in drei vSphere-Zonen gelten neben den Anforderungen für vSphere-Cluster u. a. folgende spezielle Anforderungen:

- Drei vSphere-Zonen, die über einen vSphere Distributed Switch (VDS) verbunden sind
- Jede vSphere-Zone enthält einen einzelnen vSphere-Cluster
- Ein Supervisor muss für genau drei vSphere-Zonen bereitgestellt werden
- Eine vSphere-Speicherrichtlinie ist auf der vSphere-Zone verfügbar

vSphere-Namespace

Ein vSphere-Namespace ist ein Namespace auf dem Supervisor, in dem ein oder mehrere Tanzu Kubernetes Grid-Cluster bereitgestellt werden. Für jede vSphere-Namespace konfigurieren Sie rollenbasierte Zugriffssteuerung, dauerhaften Speicher, Ressourcengrenzwerte, eine Image-Bibliothek und VM-Klassen.

TKG-Dienst

TKG-Dienst ist eine Implementierung des Open Source-Cluster-API-Projekts, das einen Satz benutzerdefinierter Ressourcen und Controller definiert, um den Lebenszyklus von Kubernetes-Clustern zu verwalten. Tanzu Kubernetes Grid ist eine Komponente von Supervisor.

TKG-Dienst stellt drei Ebenen von Controllern zur Verfügung, um den Lebenszyklus von TKG-Clustern zu verwalten. Dazu zählen VM-Dienst, Cluster-API und Cloud-Anbieter-Plug-In.

VM-Operator

Der VM-Dienst-Controller stellt eine deklarative API im Kubernetes-Stil für die Verwaltung von VMs und zugeordneten vSphere-Ressourcen bereit. Der VM-Dienst führt das Konzept „Klassen virtueller Maschinen“ ein, die abstrakte wiederverwendbare

Hardwarekonfigurationen darstellen. TKG-Dienst verwendet den VM-Dienst, um den Lebenszyklus der Steuerungsebene und der Worker-Knoten-VMs zu verwalten, die einen Arbeitslastcluster hosten.

Cluster-API

Der Cluster-API-Controller stellt deklarative APIs im Kubernetes-Stil für die Erstellung, Konfiguration und Verwaltung von Clustern bereit. Die Eingaben für die Cluster-API umfassen eine Ressource, die den Cluster beschreibt, eine Gruppe von Ressourcen, die die virtuellen Maschinen beschreiben, aus denen sich der Cluster zusammensetzt, sowie eine Reihe von Ressourcen, die Cluster-Add-Ons beschreiben.

Cloud-Anbieter-Plug-In

Der TKG-Dienst stellt Arbeitslastcluster bereit, die die Komponenten enthalten, die für die Integration der zugrunde liegenden vSphere-Namespace-Ressourcen erforderlich sind. Zu diesen Komponenten gehört ein Cloud-Anbieter-Plug-In, das in den Supervisor integriert wird. TKG verwendet das Cloud-Anbieter-Plug-In, um Anforderungen bezüglich dauerhafter Volumes an den Supervisor zu übergeben, der in den VMware Cloud Native Storage (CNS) integriert ist.

Tanzu Kubernetes-Versionen

Eine Tanzu Kubernetes-Version bietet die von VMware signierte und unterstützte Kubernetes-Softwareverteilung und Add-Ons für die Verwendung mit Tanzu Kubernetes Grid-Clustern.

Jede Tanzu Kubernetes-Version wird als VM-Vorlage (OVA-Datei) verteilt. Der Tanzu Kubernetes Grid verwendet das OVA-Format, um die Knoten der virtuellen Maschine für TKG-Cluster zu erstellen. Tanzu Kubernetes-Versionen werden entsprechend der Kubernetes-Versionsverwaltung versioniert und enthalten Betriebssystemanpassungen und Optimierungen für die vSphere-Infrastruktur.

Eine Liste der Tanzu Kubernetes-Versionen und Informationen zur Kompatibilität mit dem Supervisor finden Sie in den Versionshinweisen der Tanzu Kubernetes-Versionen. Weitere Informationen finden Sie auch in der vSphere IaaS control plane-Supportrichtlinie.

Inhaltsbibliothek

Tanzu Kubernetes-Versionen werden TKG-Clustern mithilfe einer vCenter-Inhaltsbibliothek zur Verfügung gestellt. Sie können eine abonnierte Inhaltsbibliothek erstellen und automatisch TKRs empfangen, wenn sie von VMware zur Verfügung gestellt werden, oder eine lokale Inhaltsbibliothek verwenden und TKRs manuell hochladen.

TKG-Dienst-Cluster Komponenten

Die Komponenten, die in einem TKG-Dienst-Cluster ausgeführt werden, umfassen vier Bereiche des Produkts: Authentifizierung, Speicher, Netzwerk und Lastausgleich.

Authentifizierungs-Webhook

Der Authentifizierungs-Webhook wird als Pod innerhalb des Clusters ausgeführt, um Benutzerauthentifizierungstoken zu validieren.

TKG-Cluster unterstützen die Authentifizierung auf zwei Arten: über vCenter Single Sign-On und über einen externen Identitätsanbieter, der das Open ID Connect (OIDC)-Protokoll unterstützt.

TKG führt den Pinniped-OIDC-Client auf Supervisor und auf TKG-Clusterknoten aus. Durch die Konfiguration eines externen OIDC-Anbieters für Supervisor werden die Pinniped-Komponenten automatisch konfiguriert.

TKG unterstützt verschiedene Clients für die Authentifizierung mit Supervisor. Dazu zählen das vSphere-Plug-In für kubectl und die Tanzu-CLI.

Container-Speicherschnittstelle (CSI)

Ein paravirtuelles CSI-Plug-In ist ein Kubernetes-Pod, der in einem TKG-Cluster ausgeführt wird und in VMware Cloud Native Storage (CNS) über Supervisor integriert wird. Ein Kubernetes-Pod, der in einem TKG-Cluster ausgeführt wird, kann drei Arten von virtuellen Festplatten mounten: flüchtig, dauerhaftes Volume und Container-Image.

Flüchtige Speicherung

Ein Pod benötigt flüchtigen Speicher, um flüchtige Daten wie Kubernetes-Objekte als Protokolle, Volumes und Konfigurationszuordnungen zu speichern. Die flüchtige Speicherung wird beibehalten, solange der Pod vorhanden ist. Flüchtige Daten bleiben auch nach einem Neustart des Containers erhalten, aber sobald der Pod gelöscht wird, wird die virtuelle Festplatte, auf der die flüchtigen Daten gespeichert sind, nicht mehr angezeigt.

Dauerhafte Speicherung

TKG nutzt das vSphere-Speicherrichtlinien-Framework für die Definition von Speicherklassen und die Reservierung dauerhafter Volumes. TKG-Cluster übergeben Anforderungen bezüglich dauerhafter Volumes an den Supervisor, der in den VMware Cloud Native Storage (CNS) integriert ist. Dauerhafte Volumes werden auf Clusterknoten dynamisch mithilfe einer Speicherklasse oder manuell bereitgestellt.

Container-Image-Speicherung

Container innerhalb eines Kubernetes-Pods verwenden Images, die die auszuführende Software enthalten. Der Pod mountet Images, die von ihren Containern als virtuelle Image-Festplatten verwendet werden. Wenn der Pod seinen Lebenszyklus abgeschlossen hat, werden die virtuellen Image-Festplatten vom Pod getrennt. Kubelet ist dafür verantwortlich, Container-Images aus der Image-Registrierung abzurufen und sie in virtuelle Festplatten umzuwandeln, um sie innerhalb des Pods auszuführen.

Container Network Interface (CNI)

Das Container Network Interface-Plug-In ist ein CNI-Plug-In, das Pod-Netzwerke bereitstellt.

TKG-Cluster unterstützen die folgenden Container Network Interface (CNI)-Optionen: Antrea (Standard) und Calico. Darüber hinaus stellt TKG die Antrea-NSX-geroutete CNI für die Implementierung routingfähiger Pods-Netzwerke bereit.

In der Tabelle werden Netzwerkfunktionen von TKG-Clustern und deren Implementierung zusammengefasst.

Tabelle 2-1. TKG-Dienst-Cluster-Netzwerk

Endpoint	Anbieter	Beschreibung
Pod-Konnektivität	Antrea oder Calico	Containernetzwerkschnittstelle für Pods. Antrea verwendet Open vSwitch. Calico verwendet die Linux-Bridge mit BGP.
Diensttyp: ClusterIP	Antrea oder Calico	Standardmäßiger Kubernetes-Diensttyp, auf den nur innerhalb des Clusters zugegriffen werden kann.
Diensttyp: NodePort	Antrea oder Calico	Ermöglicht externen Zugriff über einen Port, der vom Kubernetes-Netzwerk-Proxy auf jedem Worker-Knoten geöffnet wird.
Netzwerkrichtlinie	Antrea oder Calico	Steuert den Datenverkehr, der von und zu ausgewählten Pods und Netzwerk-Endpoints zulässig ist. Antrea verwendet Open vSwitch. Calico verwendet Linux-IP-Tabellen.

Cloud-Anbieterimplementierung

Mit der Cloud-Anbieterimplementierung können Sie Kubernetes-Lastausgleichsdienste und Ingress-Dienste erstellen.

Tabelle 2-2. TKG-Lastausgleich

Endpoint	Anbieter	Beschreibung
Diensttyp: LoadBalancer	Der eingebettete Lastausgleichsdienst (Teil des NSX-Netzwerk-Stacks) NSX Advanced Load Balancer (separate Installation für die Verwendung mit VDS-Netzwerk) HAProxy (separate Installation für die Verwendung mit VDS-Netzwerk)	Für den eingebetteten NSX-Lastausgleichsdienst ein virtueller Server pro Diensttypdefinition. Weitere Informationen zu NSX Advanced Load Balancer finden Sie in diesem Abschnitt dieser Dokumentation. Weitere Informationen zu HAProxy finden Sie in diesem Abschnitt dieser Dokumentation. **Hinweis** Einige Lastausgleichsfunktionen sind möglicherweise nicht für jeden unterstützten Lastausgleichstyp verfügbar, z. B. statische IPs.
Cluster-Ingress	Ingress-Controller eines Drittanbieters	Bietet Routing für eingehenden Pod-Datenverkehr. Sie können jeden Ingress-Controller von Drittanbietern verwenden, wie z. B. Contour.

TKG-Dienst-Cluster-APIs

TKG-Dienst bietet zwei APIs für die Bereitstellung und Verwaltung des Lebenszyklus von TKG-Clustern.

- API-Version `v1alpha3` für Tanzu Kubernetes-Cluster
- API-Version `v1beta1` für Cluster basierend auf einer ClusterClass

Mit der v1alpha3-API können Sie konforme Kubernetes-Cluster des Typs `TanzuKubernetesCluster` erstellen. Dieser Clustertyp ist mit gängigen Standardeinstellungen für die schnelle Bereitstellung vorkonfiguriert und kann angepasst werden. Mit der v1beta1-API können Sie basierend auf einer von VMware bereitgestellten standardmäßigen `ClusterClass` konforme Kubernetes-Cluster des Typs `Cluster` erstellen.

Hinweis Um vSphere IaaS control plane von vSphere 7 auf vSphere 8 zu aktualisieren, muss im TKG-Cluster die v1alpha2-API ausgeführt werden. Die v1alpha2-API wurde mit v7.0 Update 3 eingeführt. Die v1alpha1-API ist veraltet. Weitere Informationen finden Sie unter #unique_51.

TKG-Dienst-Cluster-Clients

TKG auf vSphere 8 Supervisor unterstützt verschiedene Clientschnittstellen für die Bereitstellung, Überwachung und Verwaltung von TKG-Clustern.

- vSphere Client zum Konfigurieren von Supervisor und Überwachen von bereitgestellten TKG-Clustern.

- vSphere-Plug-In für kubectl für die Authentifizierung mit Supervisor und TKG-Clustern mithilfe von vCenter Single Sign-On.
- kubectl zur deklarativen Bereitstellung und Verwaltung des Lebenszyklus von TKG-Clustern und zur Interaktion mit Supervisor.
- vSphere Docker Credential Helper, um Images in eine und aus einer Container-Registry zu übertragen.
- Tanzu-CLI für die Bereitstellung von Clustern mit Befehlen und für die Installation von Tanzu-Paketen.
- Tanzu Mission Control-Webschnittstelle für die Verwaltung von TKG-Clustern.

Bereitstellen von TKG-Dienst-Clustern

Für die Bereitstellung von TKG-Dienst-Clustern installieren oder aktualisieren Sie den TKG-Dienst, verwenden eine Inhaltsbibliothek zum Speichern der Tanzu Kubernetes-Versionen und konfigurieren einen oder mehrere vSphere-Namespaces für das Hosten von TKG-Clustern.

Bereitstellen von TKG-Dienstclustern

Zum Bereitstellen von TKG-Dienst-Clustern schließen Sie den Aktivierungsvorgang für die **Arbeitslastverwaltung** ab und konfigurieren eine Supervisor-Instanz.

Hinweis Eine Anleitung finden Sie unter *Installieren und Konfigurieren der vSphere IaaS-Steuerungsebene*.

Nachdem Sie **Arbeitslastverwaltung** aktiviert und Supervisor bereitgestellt haben, führen Sie die folgenden vorbereitenden Aufgaben für die Bereitstellung von TKG-Dienst-Clustern aus.

Aufgabe	Beschreibung
Installation oder Upgrade des TKG-Dienst	Weitere Informationen hierzu finden Sie unter Kapitel 3 Installieren und Aktualisieren des TKG-Diensts.
Installieren der CLIs und Herstellen einer Verbindung mit Supervisor	Weitere Informationen hierzu finden Sie unter Kapitel 4 Konfigurieren der Identität und des Zugriffs für TKG-Dienst-Cluster.
Erstellen und Synchronisieren der Inhaltsbibliothek für Tanzu Kubernetes Releases (TKrs)	Die Inhaltsbibliothek integriert die TKG-Umgebung mit dem VMware Content Delivery Network, über das die TKrs bereitgestellt werden. Sie können je nach Ihren Anforderungen eine abonnierte oder eine lokale Inhaltsbibliothek verwenden. Vorhandene TKrs müssen aktualisiert werden, um die neuen TKG-Funktionen nutzen zu können. Weitere Informationen hierzu finden Sie unter Kapitel 5 Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster.

Aufgabe	Beschreibung
Erstellen und Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienstclustern	Durch das Definieren eines vSphere-Namespace wird das Netzwerksegment für das Hosten von TKG-Dienstclustern erstellt. Weitere Informationen hierzu finden Sie unter Kapitel 6 Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern. Jeder vSphere-Namespace muss mit Folgendem konfiguriert werden: ■ Clusterbenutzer und -rollen ■ Synchronisierte TKr-Inhaltsbibliothek ■ Verknüpfte VM-Klassen ■ vSphere-Speicherrichtlinie für TKG-Dienstclusterknoten und dauerhafte Volumes Wenn Sie einen in drei vSphere-Zonen bereitgestellten Supervisor verwenden, muss in der vSphere-Speicherrichtlinie die `Zonal`-Topologie angegeben werden. Weitere Informationen hierzu finden Sie unter Erstellen einer vSphere-Speicherrichtlinie für TKG-Dienst-Cluster.
Bereitstellen von TKG-Dienst-Clustern	Weitere Informationen hierzu finden Sie unter Kapitel 7 Bereitstellen von TKG-Dienstclustern.
Betreiben von TKG-Dienst-Clustern	Weitere Informationen hierzu finden Sie unter Kapitel 8 Betreiben von TKG-Dienstclustern.
Bereitstellen von Cluster-Arbeitslasten	Weitere Informationen hierzu finden Sie unter Kapitel 12 Bereitstellen von Arbeitslasten auf TKG-Dienst-Clustern.
Wartung von TKG-Dienst-Clustern	Weitere Informationen hierzu finden Sie unter Kapitel 9 Aktualisieren von TKG-Dienstclustern.
Fehlerbehebung für TKG-Dienst-Cluster	Weitere Informationen hierzu finden Sie unter Kapitel 21 Fehlerbehebung bei TKG-Dienstclustern.

Referenzarchitekturen für TKG-Dienst-Cluster

Dieses Thema bietet eine Reihe von Referenzarchitekturen für TKG-Dienst-Cluster mit verschiedenen Bereitstellungstopologien.

TKG-Dienstcluster mit NSX-Topologie

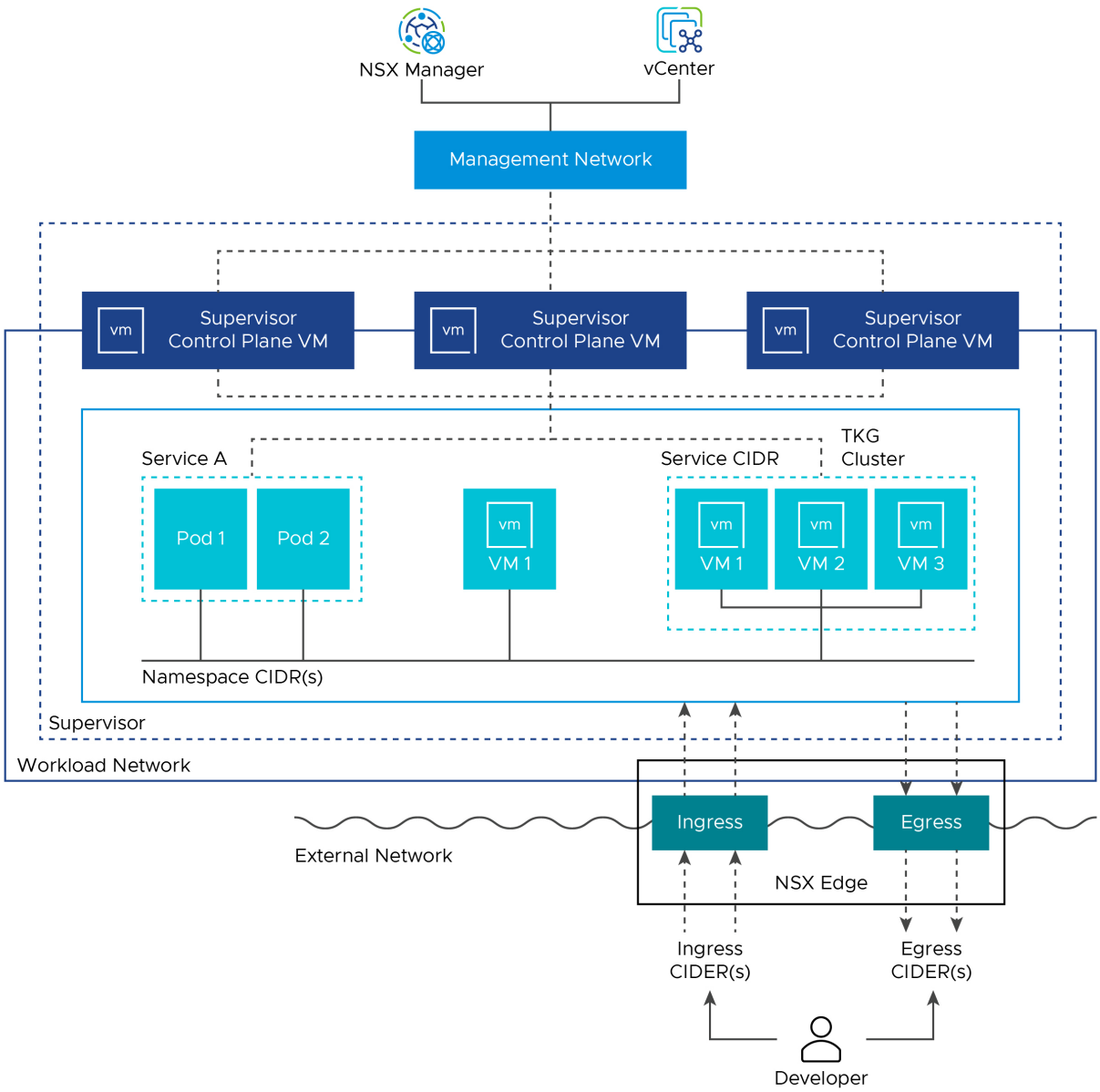
Die Referenzarchitektur veranschaulicht einen TKG-Cluster auf Supervisor mit NSX-Netzwerk. In einer solchen Umgebung hostet die Management Plane vCenter Server und NSX Manager. Die Computing-Ebene hostet NSX Edge-Knoten und Supervisor-Knoten.

Die folgenden NSX-Netzwerkobjekte sind vorhanden:

- Tier-1-Gateway (Router)
- Segment (Switch) mit dem Gateway verknüpft
- Lastausgleichsdienstserver
- Serverpool für jeden virtuellen Server der Steuerungsebene des TKG-Clusters

- Virtueller Server für jede Lastausgleichsdienstinstanz des Kubernetes-Diensts

Abbildung 2-2. TKG-Dienstcluster mit NSX-Topologie

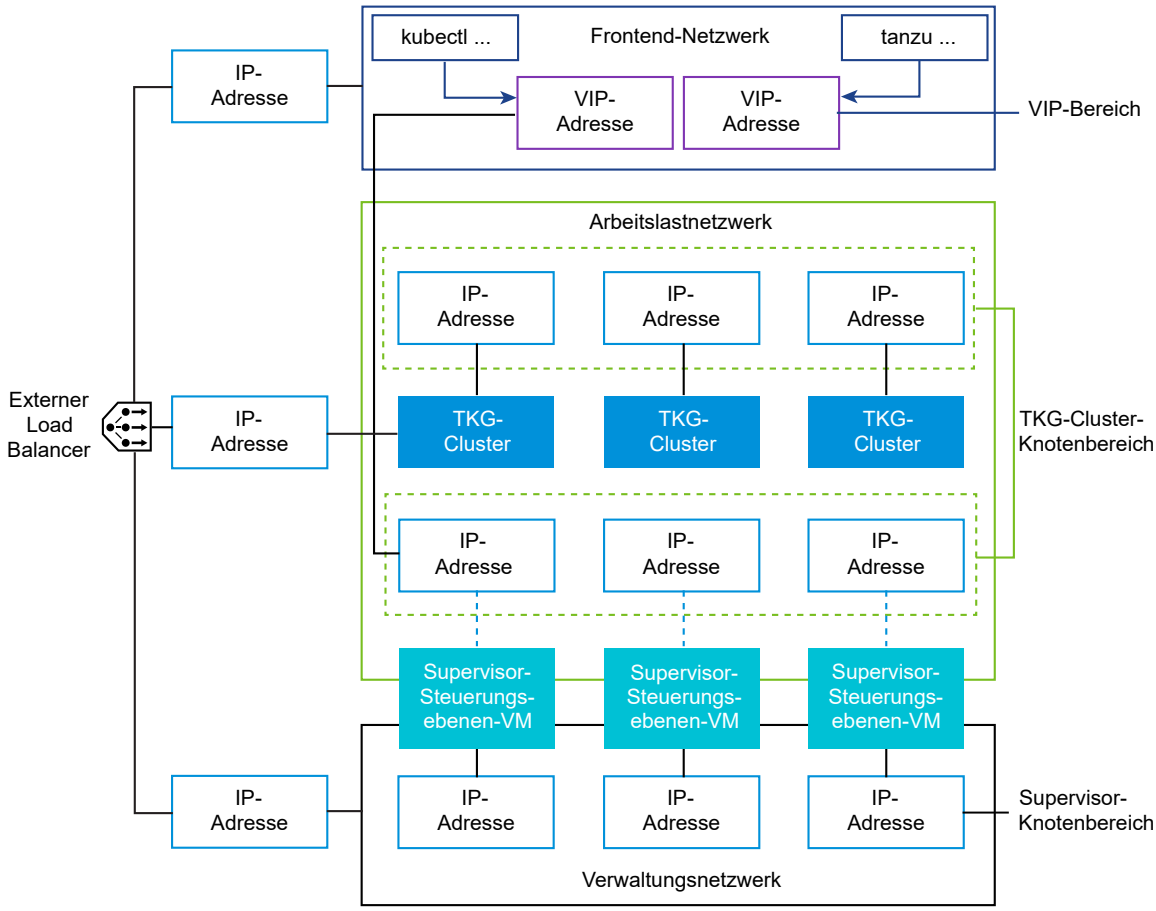


TKG-Dienstcluster mit VDS-Topologie

Die Referenzarchitektur veranschaulicht einen TKG-Dienst-Cluster mit VDS-Netzwerk und einen externen Lastausgleichsdienst. In einer solchen Umgebung sind die folgenden Netzwerke vorhanden:

- Verwaltungsnetzwerk für Supervisor-Steuerungsebenen-VMs
- Arbeitslastnetzwerk für TKG-Cluster
- Frontend-Netzwerk, über das Entwickler eine Verbindung mit TKG-Dienst-Clustern herstellen

Abbildung 2-3. TKG-Dienstcluster mit VDS-Topologie



TKG-Dienstcluster mit vSphere-Zonentopologie

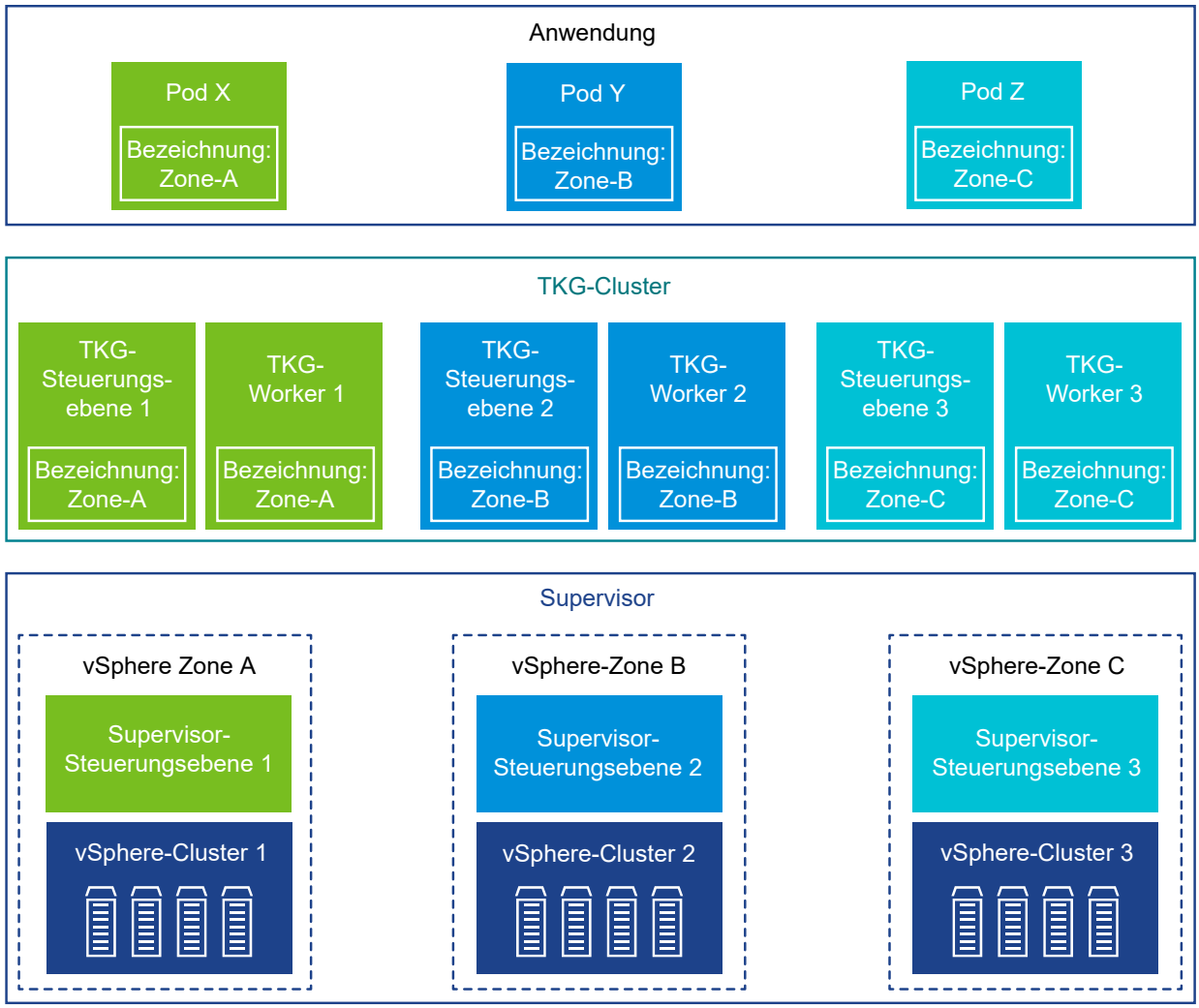
Die Referenzarchitektur stellt einen TKG-Cluster dar, der über vSphere-Zonen bereitgestellt wird. Jede vSphere-Zone wird einem vSphere-Cluster zugeordnet. Dabei handelt es sich um eine Sammlung von ESXi-Hosts, die von vCenter verwaltet und über einen vSphere Distributed Switch verbunden sind, wobei gemeinsam genutzter Speicher und Dienste aktiviert sind.

In einer Zonentopologie stellen Sie Supervisor in drei vSphere-Zonen bereit. Das System sucht eine Supervisor-Steuerungsebene in jeder vSphere-Zone und sorgt so für Hochverfügbarkeit im Falle eines Ausfalls.

Wenn Sie einen TKG-Cluster auf Supervisor bereitstellen, erkennt der Cluster die vSphere-Zonen. Die Zonentopologie unterstützt Fehlerdomänen für hochverfügbare Arbeitslasten. Bei Bedarf können Sie eine Arbeitslast in einer bestimmten Zone mithilfe von Anmerkungen ausführen.

Hinweis vSphere Zonen sind neu für vSphere 8. Daher unterstützen vSphere Zonen nur neue Bereitstellungen von Supervisor- und TKG-Clustern.

Abbildung 2-4. TKG-Dienstcluster mit vSphere-Zonentopologie



Installieren und Aktualisieren des TKG-Diensts

3

Dieser Abschnitt enthält Informationen zum Installieren und Aktualisieren des TKG-Diensts.

Lesen Sie als Nächstes die folgenden Themen:

- Verwenden der TKG-Dienst
- Überprüfen des TKG-Dienststatus
- Registrieren einer neuen Version des TKG-Diensts
- Upgrade der TKG-Dienstversion
- Fehlerbehebung beim TKG-Dienst

Verwenden der TKG-Dienst

Mit dem VMware Tanzu Kubernetes Grid-Dienst (TKG-Dienst) können Sie Kubernetes-Arbeitslastcluster auf der vSphere IaaS control plane bereitstellen. Der TKG-Dienst bietet unabhängige Versionen und asynchrone Upgrades ohne Arbeitslastunterbrechungen.

Einführung in den TKG-Dienst

Ab vSphere 8.0 Update 3 wird Tanzu Kubernetes Grid als ein Supervisor-Dienst installiert. Diese Änderung an der Architektur entkoppelt TKG von vSphere IaaS control plane-Versionen und ermöglicht Ihnen ein Upgrade des TKG-Diensts unabhängig von vCenter Server und vom Supervisor.

TKG-Dienst 3.0 wird auf den Knoten der Supervisor-Steuerungsebene installiert und ausgeführt. Der TKG-Dienst wird als verschachtelte Sammlung von Carvel-Paketen bereitgestellt. Als Kern-Supervisor-Dienst kann der TKG-Dienst auch in Umgebungen mit Internetbeschränkung aktualisiert werden. Er kann aber weder deinstalliert noch herabgestuft werden. Sie können den TKG-Dienst auf der Registerkarte **Arbeitslastverwaltung > Dienste** überwachen und verwalten. Weitere Informationen hierzu finden Sie unter Durchführen eines Upgrades der TKG-Dienst-Version.

TKG-Dienst Version 3.1 ist die erste unabhängige Version, auf die Sie ein Upgrade durchführen. Die TKG-Dienst Registrieren neuer TKG-Dienstversionen mit vCenter und das Durchführen eines Upgrades der TKG-Dienst-Version sind separate Prozesse.

Installieren von TKG-Dienst 3.0

Die Installation des TKG-Dienst erfolgt automatisch, wenn Sie alle vSphere IaaS control plane-Komponenten auf die erforderlichen Versionen aktualisieren. Weitere Informationen finden Sie in den TKG-Dienst-Versionshinweisen.

Registrieren neuer TKG-Dienstversionen mit vCenter

Das TKG-Dienstpaket wird mit vCenter Server veröffentlicht und an die öffentliche Registrierung von VMware übertragen. Die TKG-Dienstregistrierung wird auf der vCenter Server-Ebene durchgeführt. Sie haben zwei Optionen, um neue Versionen des TKG-Diensts zu registrieren: synchron und asynchron.

Tabelle 3-1. Registrierungsoptionen für TKG-Dienstversionen

Registrierungsmethode	Beschreibung
Synchron	Warten Sie auf ein Update auf die neueste vCenter Server-Version, um eine neue Version des TKG-Diensts automatisch zu registrieren. Aktualisieren Sie dann den Supervisor, um die eingebettete Registrierung mit den neuen Versionen aufzufüllen.
Asynchron	Laden Sie eine neue TKG-Dienstversionsdefinition aus der öffentlichen Registrierung herunter, und registrieren Sie sie dann manuell bei vCenter Server.

Für die synchrone Registrierung ist eine Systemaktualisierung erforderlich. Beim Aktualisieren von vCenter Server werden neue TKG-Dienstversionen automatisch bei Supervisor registriert. Zum Verwenden einer automatisch registrierten (neuen) Version müssen Sie jedoch den Supervisor auf die Version aktualisieren, die mit der von diesem vCenter Server bereitgestellten vSphere-Namespaces-Version ausgeliefert wird. Nach dem Update des Supervisors ist das Carvel-Paket für den TKG-Dienst in der eingebetteten Supervisor-Registrierung verfügbar und bereit für die Bereitstellung. Bei einem Supervisor-Upgrade wird der TKG-Dienst nicht automatisch aktualisiert. Sie müssen die gewünschte Version bereitstellen.

Die asynchrone Registrierung erfordert keine vCenter Server- und Supervisor-Updates, vorausgesetzt, die aktuelle Supervisor-Version befindet sich im Supportfenster. Die asynchrone Registrierung weist den folgenden Workflow auf:

1. Laden Sie die YAML-Datei für die Dienstdefinition von der öffentlichen Registrierungs-Site für Supervisor-Dienste herunter.
2. Registrieren Sie die neue Version des TKG-Diensts, indem Sie die Dienstdefinition auf vCenter Server hochladen.

Die Tabelle fasst die Registrierungsdetails des TKG-Diensts zusammen.

Tabelle 3-2. Registrierung der TKG-Dienstversion

TKG-Diensteigenschaft	vCenter-Paket	Öffentliche Registrierung
Registrierung neuer Versionen	Automatisch registriert	Manuelle Registrierung
Löschung neu registrierter Versionen	Nicht zulässig	Zulässig
Image-Speicherort	In die Supervisor-Steuerungsebene eingebettete Registrierung	Öffentliche Registrierung

Durchführen eines Upgrades der TKG-Dienst-Version

Upgrades der TKG-Dienstversion werden auf der Supervisor-Ebene durchgeführt. Sobald der TKG-Dienst registriert ist, aktualisieren Sie den TKG-Dienst. Stellen Sie ihn dazu als Supervisor-Dienst auf dem Ziel-Supervisor bereit. Weitere Informationen hierzu finden Sie unter Upgrade der TKG-Dienstversion.

Zum Durchführen eines Upgrades des TKG-Dienst in einer Umgebung mit beschränktem Internet („air-gapped“) registrieren Sie die neue TKG-Dienst-Version synchron, indem Sie vCenter Server aktualisieren. Wenn Sie die zu installierende Version auswählen, wird die lokale Registrierung zum Installieren der neuen TKG-Dienst-Version verwendet. Weitere Informationen hierzu finden Sie unter Registrieren neuer TKG-Dienstversionen mit vCenter.

Wenn Sie ein Upgrade der TKG-Dienstversion durchführen, führt das System Vorabprüfungen durch und meldet zwei Schweregrade:

- WARNUNG: nicht blockierend
- FEHLER: blockierend

Eine Kubernetes-Versionsüberprüfung ist ein Beispiel für eine nicht blockierende Warnungsüberprüfung. Eine Überprüfung der Supervisor-Version ist ein Beispiel für einen Blockierungsfehler. Weitere Informationen finden Sie in der Dokumentation zu Supervisor-Diensten.

Überprüfen des TKG-Dienststatus

In diesem Thema erhalten Sie Informationen zum Überprüfen des TKG-Dienststatus.

Führen Sie diese Aufgabe aus, um zu überprüfen, ob der TKG-Dienst als zentraler Supervisor-Dienst installiert ist, und um seinen Status zu überprüfen.

Es gibt zwei Möglichkeiten, den Status zu überprüfen: mit vSphere Client und mit „kubectl“. Wenn Sie den Status mit vSphere Client überprüfen möchten, melden Sie sich bei vCenter Server an, und navigieren Sie zu **Arbeitslastverwaltung > Dienste**.

Führen Sie die folgenden Schritte aus, um den Status mithilfe von „kubectl“ zu überprüfen.

Voraussetzungen

Bei dieser Aufgabe wird davon ausgegangen, dass Sie alle Systemkomponenten aktualisiert und TKG-Dienst 3.0 installiert haben. Siehe #unique_67/unique_67_Connect_42_TABLE_CED10728-0714-4D00-BF58-D4BBAA2A4D8E.

Verfahren

1 Melden Sie sich mithilfe von kubectl bei Supervisor an.

```
kubectl vsphere login --server=<SUPERVISOR-IP-or-FQDN> --vsphere-username <VCENTER-SSO-USER>
```

2 Führen Sie den folgenden Befehl aus.

```
kubectl get packageinstall --namespace vmware-system-supervisor-services
```

Sie sollten sehen, dass der TKG-Dienst installiert ist.

```
NAMESPACE                          NAME                        PACKAGE NAME
PACKAGE VERSION  DESCRIPTION          AGE
vmware-system-supervisor-services   svc-
tkg.vsphere.vmware.com  tkg.vsphere.vmware.com  0.0.1-b836be7    Reconcile succeeded  17h
```

Registrieren einer neuen Version des TKG-Diensts

In diesem Thema finden Sie Informationen zum manuellen Registrieren einer neuen Version des TKG-Diensts bei vCenter Server zum Durchführen des asynchronen Upgrades des TKG-Diensts.

Diese Aufgabe ist nur erforderlich, wenn Sie die TKG-Dienstversion asynchron aktualisieren möchten. Weitere Informationen hierzu finden Sie unter Registrieren neuer TKG-Dienstversionen mit vCenter.

Voraussetzungen

Bei dieser Aufgabe wird davon ausgegangen, dass Sie alle Systemkomponenten aktualisiert und TKG-Dienst 3.0 installiert haben. Siehe Installieren von TKG-Dienst 3.0.

Verfahren

1 Navigieren Sie in einem Browser zur folgenden Verteilungs-Site der Supervisor-Dienste: https://www.vmware.com/go/supervisor-service.

2 Laden Sie die Datei TKG-Dienst `package.yaml` von der Site herunter.

3 Melden Sie sich mit vSphere Client bei vCenter Server an.

4 Navigieren Sie zu **Arbeitslastverwaltung > Dienste**.

5 Suchen Sie die Dienstkachel **Tanzu Kubernetes Grid Service**.

6 Wählen Sie **Aktionen > Neue Version hinzufügen** aus.

7 Klicken Sie auf **Hochladen**.

8 Wählen Sie die Datei `package.yaml` aus, die Sie heruntergeladen haben.

9 Klicken Sie auf **Fertigstellen**.

Ergebnisse

Nach der Registrierung der neuen Dienstdefinition kann es sein, dass mehrere Versionen des TKG-Diensts für die Kachel TKG-Dienst als verfügbar angezeigt werden. Sie wählen die Zielversion aus, wenn Sie ein Upgrade von TKG-Dienst durchführen.

Nächste Schritte

Upgrade der TKG-Dienstversion.

Upgrade der TKG-Dienstversion

Informationen zum Upgrade der Version des TKG-Diensts finden Sie in diesem Thema.

Führen Sie die folgenden Schritte aus, um die TKG-Dienstversion zu aktualisieren.

Das Upgrade der TKG-Dienstversion wird auf der Supervisor-Ebene durchgeführt. Wenn vCenter Server mehrere Supervisoren hostet, müssen Sie den Ziel-Supervisor auswählen.

Voraussetzungen

Bei dieser Aufgabe wird davon ausgegangen, dass Sie alle Systemkomponenten aktualisiert und TKG-Dienst 3.0 installiert haben. Siehe Installieren von TKG-Dienst 3.0.

Bei dieser Aufgabe wird davon ausgegangen, dass Sie Registrieren einer neuen Version des TKG-Diensts oder Registrieren neuer TKG-Dienstversionen mit vCenter eine neue Version des TKG-Diensts registriert haben.

Verfahren

1 Melden Sie sich mit vSphere Client bei vCenter Server an.

2 Navigieren Sie zu **Arbeitslastverwaltung > Dienste**.

3 Suchen Sie die Kachel **Konfiguration des Tanzu Kubernetes Grid Service**.

4 Wählen Sie **Aktionen > Auf Supervisoren installieren** aus.

5 Wählen Sie die Zielversion des TKG-Diensts aus, auf die ein Upgrade durchgeführt werden soll.

6 Wählen Sie den Ziel-Supervisor aus, der den TKG-Dienst hosten soll, wofür Sie ein Upgrade durchführen möchten.

7 Bestätigen Sie die Dienstkompatibilität, und klicken Sie auf **OK**.

8 Bestätigen Sie, dass der TKG-Dienst aktualisiert wird.

   Auf der Kachel **Tanzu Kubernetes Grid Service** auf der Seite **Arbeitslastverwaltung > Dienste** werden die Version und der Status angezeigt. Sie können den Status auch mithilfe von `kubectl get tkr` überprüfen.

Fehlerbehebung beim TKG-Dienst

In diesem Thema finden Sie Informationen zur Fehlerbehebung beim TKG-Dienst.

TKG-Dienst-Support-Paket

Das TKG-Dienst-Support-Paket ist im Supervisor-Support-Paket enthalten. Weitere Informationen finden Sie unter Erfassen eines Support-Pakets für Supervisor.

Innerhalb des Supervisor-Support-Pakets befinden sich die Protokolle des TKG-Diensts im Ordner `var/log/tkg-svs`.

Supervisor-Dienste- und vom zentralen Services Controller verwaltete Protokolle für den zentralen Services Controller befinden sich unter `/var/log/vmware/wcp/`.

Anwendungsplattformprotokolle können mithilfe des folgenden Befehls abgerufen werden.

```
kubectl logs vmware-system-appplatform-lifecycle-xxx -n vmware-system-appplatform-operator-system
```

Konfigurieren der Identität und des Zugriffs für TKG-Dienst-Cluster

4

Dieser Abschnitt enthält Informationen zur Konfiguration der Benutzerauthentifizierung und -autorisierung für TKG-Dienst-Cluster auf Supervisor.

Lesen Sie als Nächstes die folgenden Themen:

- Informationen zur Identitäts- und Zugriffsverwaltung für TKG-Dienst-Cluster
- Installieren von CLI-Tools für TKG-Dienst-Cluster
- Herstellen einer Verbindung zu TKG-Dienst-Clustern mithilfe der vCenter SSO-Authentifizierung
- Herstellen einer Verbindung zu TKG-Clustern auf Supervisor mithilfe eines externen Identitätsanbieters
- Herstellen einer Verbindung zu TKG-Dienst-Clustern als Kubernetes-Administrator und Systembenutzer
- Erstellen einer dedizierten Gruppe und Rolle für Plattformoperatoren

Informationen zur Identitäts- und Zugriffsverwaltung für TKG-Dienst-Cluster

DevOps-Ingenieure stellen eine Verbindung zur vSphere IaaS control plane her, um TKG-Dienst-Cluster bereitzustellen und deren Lebenszyklus zu verwalten. Entwickler stellen eine Verbindung zu TKG-Dienst-Clustern her, um Pakete, Arbeitslasten und Dienste bereitzustellen. Administratoren benötigen möglicherweise direkten Zugriff auf TKG-Dienst-Clusterknoten, um Fehler zu beheben. Auf der Plattform werden Tools für die Identitäts- und Zugriffsverwaltung sowie Methoden bereitgestellt, die alle Anwendungsfälle und Rollen unterstützen.

TKG-Dienst-Clusterzugriff erfolgt über den vSphere-Namespace

Sie stellen TKG-Dienst-Cluster in einem vSphere-Namespace bereit. Wenn Sie einen vSphere-Namespace konfigurieren, legen Sie dessen DevOps-Berechtigungen fest, einschließlich der Identitätsquelle, der Benutzer und Gruppen sowie der Rollen. Die Plattform gibt diese Berechtigungen an jeden in diesem vSphere-Namespace bereitgestellten TKG-Dienst-Cluster weiter. Die Plattform unterstützt zwei Authentifizierungsmethoden: vCenter Single Sign-On und einen OIDC-konformen externen Identitätsanbieter.

Authentifizierung mithilfe von vCenter Single Sign-On und Kubectl

Standardmäßig wird vCenter Single Sign-On zur Authentifizierung bei der Umgebung verwendet, einschließlich Supervisor- und TKG-Dienst-Clustern. vCenter Single Sign-On bietet Authentifizierung für die vSphere-Infrastruktur und kann in AD/LDAP-Systeme integriert werden. Weitere Informationen finden Sie unter vSphere-Authentifizierung mit vCenter Single Sign-On.

Für die Authentifizierung mit vCenter Single Sign-On verwenden Sie das vSphere-Plug-In für kubectl. Nach der Authentifizierung verwenden Sie kubectl, um den Lebenszyklus von TKG-Dienstclustern deklarativ bereitzustellen und zu verwalten und mit Supervisor zu interagieren.

Das vSphere-Plug-In für kubectl-Plug-In hängt von kubectl ab. Wenn Sie sich mit dem Befehl `kubectl vsphere login` authentifizieren, gibt das Plug-In eine POST-Anforderung mit Standardauthentifizierung für einen /wcp/login-Endpoint auf Supervisor aus. vCenter Server gibt ein JSON-Web-Token (JWT) aus, dem der Supervisor vertraut.

Informationen zum Herstellen einer Verbindung über vCenter Single Sign-On finden Sie unter Herstellen einer Verbindung zu TKG-Dienst-Clustern mithilfe der vCenter SSO-Authentifizierung.

Authentifizierung mithilfe eines externen Identitätsanbieters und der Tanzu-CLI

Sie können Supervisor mit einem externen Identitätsanbieter konfigurieren, der das OpenID Connect-Protokoll unterstützt. Nach der Konfiguration fungiert Supervisor als OAuth 2.0-Client und verwendet den Authentifizierungsdienst Pinniped, um mithilfe der Tanzu-CLI Client-Konnektivität bereitzustellen. Die Tanzu-CLI unterstützt die Bereitstellung und Verwaltung des Lebenszyklus von TKG-Dienst-Clustern. Jede Supervisor-Instanz kann nur einen externen Identitätsanbieter unterstützen.

Sobald das Authentifizierungs-Plug-In und der OIDC-Aussteller ordnungsgemäß für die Funktion der pinniped-auth-CLI konfiguriert sind, sucht das System bei Supervisor mit `tanzu login --endpoint` nach einigen bekannten Configmaps, um die `pinniped config`-Konfigurationsdatei zu erstellen.

Informationen zum Herstellen einer Verbindung mithilfe eines externen OIDC-Anbieters finden Sie unter Herstellen einer Verbindung zu TKG-Clustern auf Supervisor mithilfe eines externen Identitätsanbieters.

Authentifizierung mit einem hybriden Ansatz: vCenter SSO mit Tanzu CLI

Wenn Sie vCenter Single Sign-On als Identitätsanbieter verwenden und die Tanzu-CLI einsetzen möchten, können Sie einen hybriden Ansatz nutzen und sich mithilfe der beiden Tools bei Supervisor anmelden. Dieser Ansatz kann für die Installation von Standardpaketen nützlich sein. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu Supervisor mithilfe der Tanzu-CLI und der vCenter SSO-Authentifizierung.

Benutzer und Gruppen für DevOps

Die beim Konfigurieren eines vSphere-Namespace eingerichteten Berechtigungen gelten für DevOps-Benutzer, die den Lebenszyklus von TKG-Dienst-Clustern verwalten. Der DevOps-Benutzer oder die DevOps-Gruppe, der Sie Berechtigungen zuweisen, muss in der Identitätsquelle vorhanden sein. DevOps-Benutzer authentifizieren sich mit ihren Identitätsanbieter-Anmeldedaten.

DevOps-Benutzer sind dafür verantwortlich, nachgelagerten Benutzern Clusterzugriff zu gewähren, wie z. B. Entwicklern, die Arbeitslasten auf bereitgestellten Clustern zur Verfügung stellen möchten. Diese Benutzer authentifizieren sich mithilfe eines Identitätsanbieters oder einer Clusterrolle und Bindung bei Kubernetes. Dieser Vorgang wird im folgenden Abschnitt genauer beschrieben.

Hinweis vSphere-Namespace-Berechtigungen gelten ausschließlich für DevOps-Benutzer, die TKG-Dienst-Cluster erstellen und verwalten müssen. Benutzer dieser Cluster, wie z. B. Entwickler, verwenden den Kubernetes-Authentifizierungsmechanismus.

Rollenberechtigungen und -bindungen

Zwei Arten von RBAC-Systemen (Role Based Access Control, Rollenbasierte Zugriffssteuerung) stehen für TKGS-Cluster zur Verfügung: vSphere-Namespace-Berechtigungen und Kubernetes RBAC-Autorisierung. Als vSphere-Administrator weisen Sie vSphere-Namespace-Berechtigungen zu, damit Benutzer TKG-Dienstcluster erstellen und betreiben können. Cluster-Operatoren verwenden Kubernetes RBAC, um Entwicklern Clusterzugriff zu gewähren und Rollenberechtigungen zuzuweisen. Weitere Informationen hierzu finden Sie unter Gewähren von vCenter SSO-Zugriff auf TKG-Dienst-Cluster für Entwickler.

vSphere-Namespaces unterstützen drei Rollen: **Schreibzugriff**, **Lesezugriff** und **Besitzer**. Rollenberechtigungen werden zum vSphere-Namespace, in dem ein TKG-Dienst-Cluster gehostet wird, zugewiesen und auf diesen beschränkt. Weitere Informationen hierzu finden Sie unter Kapitel 6 Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern.

Ein Benutzer / eine Gruppe, dem/der die Rollenberechtigung **Kann bearbeiten** für einen vSphere-Namespace zugewiesen wurde, kann TKG-Dienst-Cluster in diesem vSphere-Namespace erstellen, lesen, aktualisieren und löschen. Wenn Sie der Rolle **Schreibzugriff** darüber hinaus einen Benutzer/eine Gruppe zuweisen, erstellt das System in jedem Cluster in diesem vSphere-

Namespace eine Rollenbindung und bindet die Berechtigung an die Kubernetes-Clusterrolle mit dem Namen `cluster-admin`. Mithilfe der Rolle `cluster-admin` können Benutzer TKG-Dienstcluster im Ziel-vSphere-Namespace bereitstellen und betreiben. Sie können diese Zuordnung mit dem Befehl `kubectl get rolebinding` im Ziel-vSphere-Namespace anzeigen.

```
kubectl get rolebinding -n tkgs-cluster-namespace
NAME                                                        ROLE
AGE
wcp:tkg-cluster-namespace:group:vsphere.local:administrators   ClusterRole/edit
33d
wcp:tkg-cluster-namespace:user:vsphere.local:administrator     ClusterRole/edit
33d
```

Ein Benutzer oder eine Gruppe, der die Rollenberechtigung **Lesezugriff** für einen vSphere-Namespace gewährt wird, hat Lesezugriff auf TKG-Dienst-Clusterobjekte, die in diesem vSphere-Namespace bereitgestellt werden. Anders als bei der Berechtigung **Schreibzugriff** wird für die Rolle **Lesezugriff** keine Kubernetes-Rollenbindung in TKGS-Clustern in diesem vSphere-Namespace erstellt. Der Grund hierfür besteht darin, dass in Kubernetes keine vergleichbare schreibgeschützte Rolle vorhanden ist, an die ein Benutzer oder eine Gruppe mit der Berechtigung **Lesezugriff** gebunden werden kann. Für andere Benutzer als `cluster-admin` verwenden Sie Kubernetes RBAC, um Zugriff zu gewähren. Weitere Informationen finden Sie unter Gewähren von vCenter SSO-Zugriff auf TKG-Dienst-Cluster für Entwickler.

Ein Benutzer/eine Gruppe, dem/der die Berechtigung **Besitzer** für einen vSphere-Namespace gewährt wurde, kann TKG-Dienst-Cluster in diesem vSphere-Namespace verwalten und zusätzliche vSphere-Namespaces mithilfe von kubectl erstellen und löschen. Wenn Sie der Rolle „Besitzer“ einen Benutzer bzw. eine Gruppe zuweisen, erstellt das System ein ClusterRoleBinding-Objekt und ordnet es einem ClusterRole-Objekt zu, mit dem der Benutzer bzw. die Gruppe vSphere-Namespaces mithilfe von kubectl erstellen und löschen kann. Sie können diese Zuordnung nicht mit demselben Verfahren anzeigen. Sie müssen sich per SSH bei einem Supervisor-Knoten anmelden, um diese Zuordnung anzuzeigen.

Hinweis Die Besitzerrolle wird für Benutzer unterstützt, die aus der vCenter Single Sign-On-Identitätsquelle stammen. Sie können die Rolle „Besitzer“ nicht mit einem Benutzer / einer Gruppe eines externen Identitätsanbieters verwenden.

vSphere-Berechtigungen

Die Tabelle zeigt die Typen von vSphere-Berechtigungen, die für verschiedene vSphere IaaS control plane-Personas erforderlich sind. Bei Bedarf können Sie eine benutzerdefinierte vSphere SSO-Gruppe und -Rolle für die Arbeitslastverwaltung erstellen. Weitere Informationen hierzu finden Sie unter Erstellen einer dedizierten Gruppe und Rolle für Plattformoperatoren.

Tabelle 4-1. vSphere-Berechtigungen für vSphere with Tanzu-Personas

Persona	vSphere-Rolle	vSphere-SSO-Gruppe	vSphere-Namespace
VI/Cloud-Administrator	Administrator	Administratoren	SSO-Benutzer und/oder AD-Benutzer
DevOps-/Plattform-Operator	Nicht-Admin oder benutzerdefinierte Rolle	ServiceProviderUsers	SSO-Benutzer und/oder AD-Benutzer
Entwickler	Schreibgeschützt oder keine	Keine	SSO-Benutzer und/oder AD-Benutzer

Systemadministrator-Konnektivität

Administratoren können als `kubernetes-admin`-Benutzer eine Verbindung mit TKG-Dienst-Clusterknoten herstellen. Diese Methode kann empfehlenswert sein, wenn die vCenter Single Sign-On-Authentifizierung nicht verfügbar ist. Zu Fehlerbehebungszwecken können Systemadministratoren mithilfe von SSH und einem privaten Schlüssel als `vmware-system-user` eine Verbindung mit einem TKG-Dienst herstellen. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu TKG-Dienst-Clustern als Kubernetes-Administrator und Systembenutzer.

Installieren von CLI-Tools für TKG-Dienst-Cluster

Die vSphere IaaS control plane bietet verschiedene CLI-Tools zur Herstellung der Verbindung mit und Verwaltung des Lebenszyklus von TKG-Dienst-Clustern.

Installieren des Kubernetes-CLI-Tools für vSphere

vCenter Single Sign-On-Benutzer können die Kubernetes-CLI-Tools für vSphere verwenden, um eine Verbindung mit TKG-Dienst-Clustern herzustellen und mit ihnen zu interagieren.

Informationen zu Kubernetes-CLI-Tools für vSphere

Das Downloadpaket für Kubernetes-CLI-Tools für vSphere (`vsphere-plugin.zip`) enthält zwei ausführbare Dateien: kubectl und das vSphere-Plug-In für kubectl.

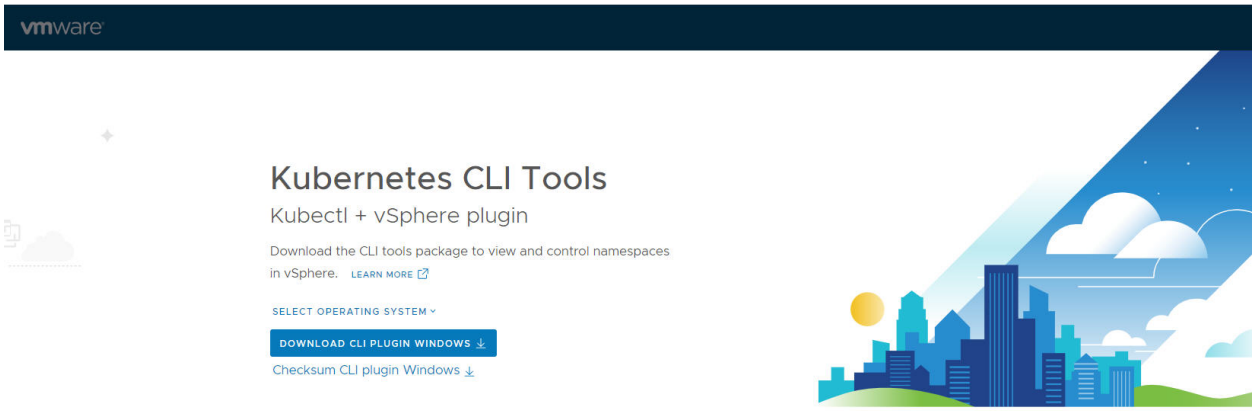
Die kubectl-CLI weist eine austauschbare Architektur (Pluggable Storage Architecture, PSA) auf. Durch das vSphere-Plug-In für kubectl werden die für kubectl verfügbaren Befehle erweitert, sodass Sie mithilfe von vCenter Single Sign-On eine Verbindung mit dem Supervisor und den TKG-Clustern herstellen können.

Hinweis Die vSphere IaaS control plane stellt Binärdateien für x86/64-Prozessoren bereit.

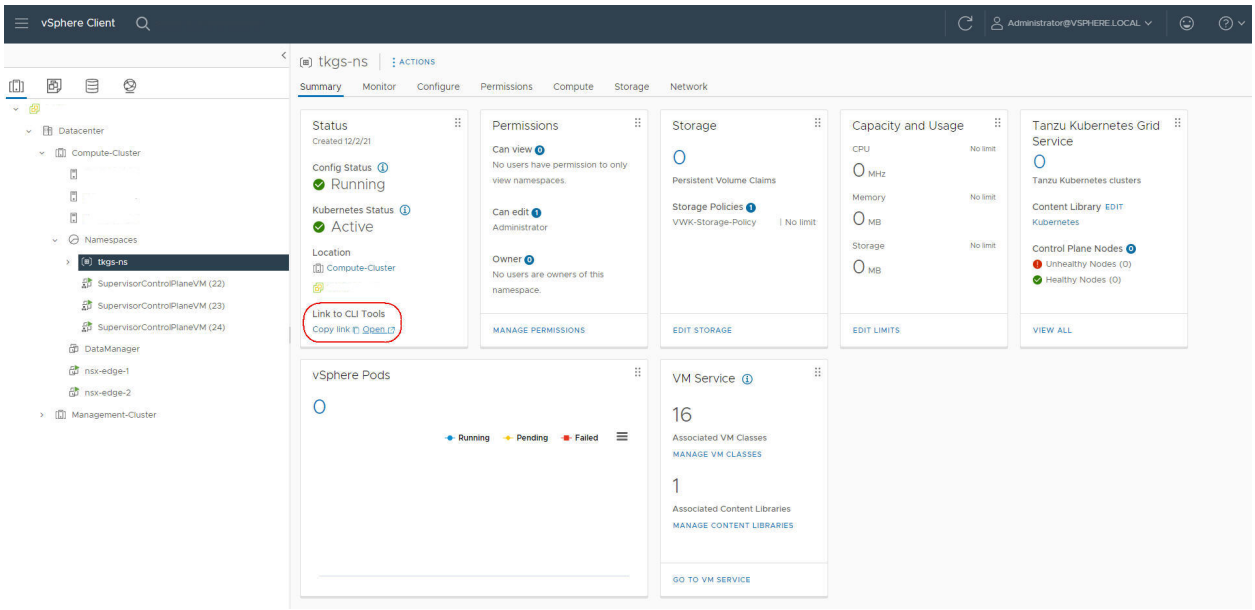
Voraussetzung: vSphere-Namespace ist erstellt

Die Kubernetes-CLI-Tools für vSphere stehen auf der vSphere IaaS control plane-DevOps-Seite zur Verfügung.

Abbildung 4-1. DevOps-Seite für vSphere



Auf die vSphere IaaS control plane-DevOps-Seite können Sie über den vSphere-Namespace-Konfigurationsbereich zugreifen. Weitere Informationen finden Sie unter Kapitel 6 Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern.



Installieren der Kubernetes-CLI-Tools für vSphere mithilfe des vSphere Client

Führen Sie die folgenden Schritte aus, um Kubernetes-CLI-Tools für vSphere auf der DevOps-Seite in vCenter zu installieren.

1 Bitten Sie den vSphere-Administrator um den Link für die Downloadseite der Kubernetes-CLI-Tools. Alternativ können Sie den Link wie folgt abrufen, falls Sie Zugriff auf vCenter Server haben:

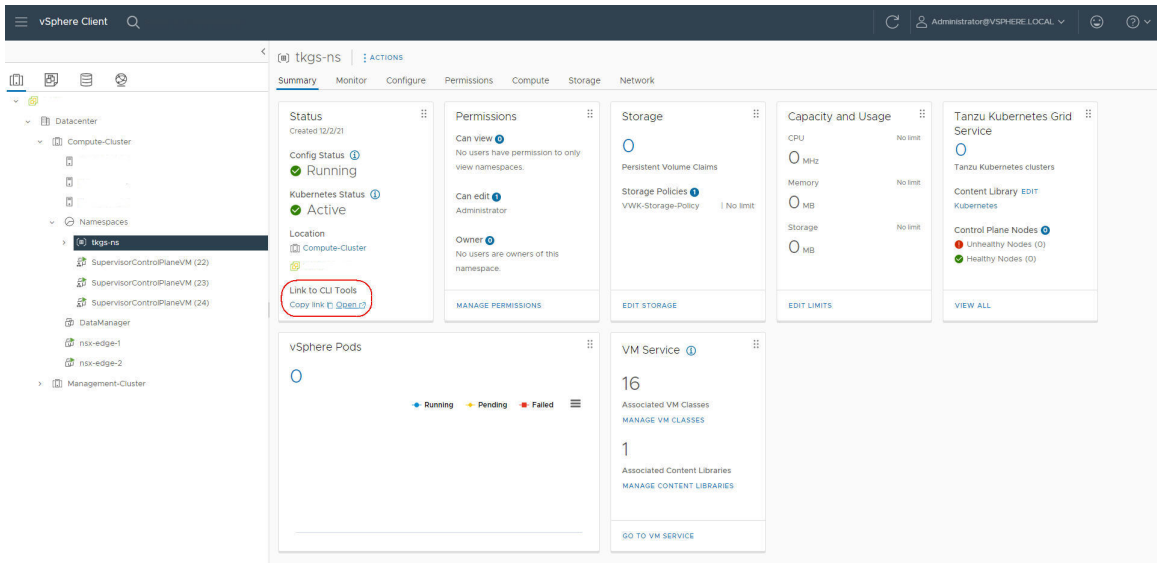
   a Melden Sie sich über vSphere Client bei vCenter Server an.

   b Navigieren Sie zu **Arbeitslastverwaltung > Namespaces**.

   c Wählen Sie den vSphere-Namespace aus, in dem Sie arbeiten.

d Wählen Sie die Registerkarte **Übersicht** aus und suchen Sie den Bereich **Status** auf dieser Seite. (Weitere Informationen finden Sie in der Abbildung.)

e Wählen Sie unter der Überschrift **Link zu CLI-Tools** die Option **Öffnen** aus, um die Downloadseite zu öffnen. Sie können den Link auch **kopieren**.



2 Wählen Sie das Betriebssystem aus.

Hinweis Lesen Sie bei Bedarf die betriebssystemspezifischen Installationsschritte am Ende dieses Abschnitts.

3 Laden Sie die Datei `vsphere-plugin.zip` herunter.

4 Extrahieren Sie den Inhalt der ZIP-Datei in ein Arbeitsverzeichnis.

5 Fügen Sie den Speicherort der beiden ausführbaren Dateien der Variable PATH Ihres Systempfads hinzu.

6 Um die Installation der kubectl-CLI zu überprüfen, starten Sie eine Sitzung über eine Shell, ein Terminal oder eine Eingabeaufforderung und führen Sie den `kubectl`-Befehl aus.

Die kubectl-Banner-Meldung und die Liste der Befehlszeilenoptionen werden für die CLI angezeigt.

7 Führen Sie zum Überprüfen der Installation des vSphere-Plug-In für kubectl den Befehl `kubectl vsphere` aus.

Die vSphere-Plug-In für kubectl-Banner-Meldung und die Liste der Befehlszeilenoptionen für das Plug-In werden angezeigt.

Installieren der Kubernetes-CLI-Tools für vSphere über die Befehlszeile

Bei Verwendung von Linux oder MacOS können Sie folgenden Befehl zum Herunterladen von Kubernetes-CLI-Tools für vSphere ausführen.

1 Melden Sie sich über vSphere Client bei vCenter Server an.

2 Wählen Sie **Arbeitslastverwaltung > Supervisor** aus.

3 Rufen Sie die IP-Adresse des Lastausgleichsdiensts für die Supervisor-Steuerungsebene ab.

4 Erstellen Sie eine entsprechende Variable.

5 Installieren Sie die Tools mithilfe des folgenden Befehls. (Möglicherweise müssen Sie sie für Ihre Umgebung anpassen.)

   Linux:

   ```
   curl -LOk https://${SC_IP}/wcp/plugin/linux-amd64/vsphere-plugin.zip
   unzip vsphere-plugin.zip
   mv -v bin/* /usr/local/bin/
   ```

   MacOS:

   ```
   curl -LOk https://${SC_IP}/wcp/plugin/macos-darwin64/vsphere-plugin.zip
   unzip vsphere-plugin.zip
   mv -v bin/* /usr/local/bin/
   ```

6 Führen Sie die Befehle `kubectl` und `kubectl vsphere` aus und überprüfen Sie die Installation.

Installieren der Kubernetes-CLI-Tools für vSphere unter Linux

Installieren Sie die Kubernetes-CLI-Tools für vSphere auf einem Linux-Host.

1 Laden Sie die Datei `vsphere-plugin.zip` für Linux herunter.

2 Extrahieren Sie den aus zwei ausführbaren Dateien bestehenden Inhalt des Archivs: kubectl und `kubectl-vsphere`.

3 Fügen Sie die beiden ausführbaren Dateien in den Suchpfad der ausführbaren Datei Ihres Betriebssystems ein, z. B. `/usr/local/bin`.

4 Führen Sie den Befehl `kubectl vsphere` aus, um die Installation zu überprüfen.

5 Führen Sie den Befehl `kubectl vsphere login --server=Supervisor-IP` aus, um sich bei Supervisor anzumelden.

6 Führen Sie den Befehl `kubectl config get-contexts` aus, um eine Liste Ihrer vSphere-Namespaces anzuzeigen, auf die Sie Zugriff haben.

7 Führen Sie den Befehl `kubectl config use-context CONTEXT` aus, um Ihren Standardkontext auszuwählen.

Installieren der Kubernetes-CLI-Tools für vSphere unter MacOS

Installieren Sie die Kubernetes-CLI-Tools für vSphere auf einem MacOS-Host.

1 Laden Sie die Datei `vsphere-plugin.zip` für MacOS herunter.

2 Extrahieren Sie den aus zwei ausführbaren Dateien bestehenden Inhalt des Archivs: kubectl und `kubectl-vsphere`.

3 Fügen Sie die beiden ausführbaren Dateien in den Suchpfad der ausführbaren Datei Ihres Betriebssystems ein, z. B. `/usr/local/bin`.

4 Führen Sie den Befehl `kubectl vsphere` aus, um die Installation zu überprüfen.

5 Führen Sie den Befehl `kubectl vsphere login --server=Supervisor-IP` aus, um sich bei Supervisor anzumelden.

6 Führen Sie den Befehl `kubectl config get-contexts` aus, um eine Liste der vSphere-Namespaces anzuzeigen, auf die Sie Zugriff haben.

7 Führen Sie den Befehl `kubectl config use-context CONTEXT` aus, um Ihren Standardkontext auszuwählen.

Installieren der Kubernetes-CLI-Tools für vSphere unter Windows

Installieren Sie die Kubernetes-CLI-Tools für vSphere auf einem Windows-Host.

1 Laden Sie die Datei `vsphere-plugin.zip` für MacOS herunter.

2 Extrahieren Sie den aus zwei ausführbaren Dateien bestehenden Inhalt des Archivs: `kubectl.exe` und `kubectl-vsphere.exe`.

3 Fügen Sie beide ausführbare Dateien in den Systempfad ein.

4 Führen Sie den Befehl `kubectl vsphere` aus, um die Installation zu überprüfen.

5 Führen Sie den Befehl `kubectl vsphere login --server=Supervisor-IP` aus, um sich bei Supervisor anzumelden.

6 Führen Sie den Befehl `kubectl config get-contexts` aus, um eine Liste der vSphere-Namespaces anzuzeigen, auf die Sie Zugriff haben.

7 Führen Sie den Befehl `kubectl config use-context CONTEXT` aus, um Ihren Standardkontext auszuwählen.

Installieren der Tanzu-CLI zur Verwendung mit TKG-Dienst-Clustern

Sie können die Tanzu-CLI verwenden, um den Lebenszyklus von TKG-Dienst-Clustern bereitzustellen und zu verwalten.

Installieren Sie die Tanzu-CLI für die Arbeit mit TKG-Dienst-Clustern.

Verfahren

1 Installationsanweisungen finden Sie in der Produktdokumentation zur Tanzu-CLI.

2 Überprüfen Sie die Kompatibilitätsmatrix auf die entsprechende Version.

Hinweis Verwenden Sie für TKG-Dienst 3.0 keine Tanzu-CLI-Version, die größer als v1.1.0 ist. Verwenden Sie beispielsweise nicht die Tanzu-CLI v1.2.0.

Hinweis Laden Sie ab TKG-Dienst 3.1.0 das CLI-Plug-in „pinniped-auth“ mit derselben Version des TKG-Diensts herunter, und laden Sie das CLI-Plug-in „imgpkg“ mit der neuesten Version herunter.

3 Installieren Sie die Tanzu-CLI, indem Sie die Anweisungen für die Zielversion befolgen.

Beispiel: Installieren und Verwenden der VMware Tanzu-CLI v1.1.x: https://docs.vmware.com/de/VMware-Tanzu-CLI/1.1/tanzu-cli/index.html

Installieren des vSphere Docker Credential Helper

Verwenden Sie die vSphere Docker Credential Helper-CLI, um Container-Images sicher an die eingebettete Harbor-Registrierung weiterzugeben und Container-Images aus der eingebetteten Harbor-Registrierung zu ziehen.

Verwenden Sie den vSphere Docker Credential Helper, um Ihren Docker-Client sicher mit der Containerregistrierung zu verbinden.

Hinweis Der vSphere Docker Credential Helper ist für die Verwendung mit der eingebetteten Harbor-Registrierung vorgesehen, die veraltet ist. Wenn Sie den Harbor-Supervisor-Dienst verwenden, machen Sie sich mit folgendem Thema vertraut, um mithilfe von Docker eine Verbindung zu diesem Dienst herzustellen: Konfigurieren eines Docker-Clients mit dem Harbor-Registrierungszertifikat.

Voraussetzungen

Die Downloadseite zu Kubernetes CLI Tools enthält einen Link zum Herunterladen des vSphere Docker Credential Helper.

- Konfigurieren Sie einen Docker-Client.
- Erhalten Sie Zugriff auf die eingebettete Harbor-Registrierung, die auf Supervisor aktiviert ist.
- Bitten Sie den vSphere-Administrator um den Link für die Downloadseite zu Kubernetes-CLI-Tools für vSphere.
- Alternativ können Sie den Link wie folgt abrufen, falls Sie Zugriff auf vCenter Server haben:
  - Melden Sie sich über vSphere Client bei vCenter Server an.
  - Navigieren Sie zu **Arbeitslastverwaltung > Namespaces** und wählen Sie den zielseitigen vSphere-Namespace aus.
  - Wählen Sie die Registerkarte **Übersicht** aus und suchen Sie nach der Kachel **Status**.
  - Wählen Sie unter der Überschrift **Link zu CLI-Tools** die Option **Öffnen** aus, um die Downloadseite zu öffnen. Sie können den Link auch **kopieren**.

Verfahren

1 Navigieren Sie mit einem Browser zur Download-URL der **Kubernetes-CLI-Tools** für Ihre Umgebung.

2 Scrollen Sie nach unten zum Abschnitt vSphere Docker Credential Helper.

3 Wählen Sie das Betriebssystem aus.

4 Laden Sie die Datei `vsphere-docker-credential-helper.zip` herunter.

5 Extrahieren Sie den Inhalt der ZIP-Datei in ein Arbeitsverzeichnis.

Die ausführbare Binär-Datei `docker-credential-vsphere` ist verfügbar.

6 Kopieren Sie die Binärdatei `docker-credential-vsphere` auf Ihren Docker-Clienthost.

7 Fügen Sie den Speicherort der Binärdatei zu Ihrem Systempfad (PATH) hinzu.

Beispielsweise unter Linux:

```
mv docker-credential-vsphere /usr/local/bin/docker-credential-vsphere
```

8 Überprüfen Sie die Installation des vSphere Docker Credential Helper, indem Sie den Befehl `docker-credential-vsphere` in einer Shell oder Terminalsitzung ausführen.

Die Banner-Meldung und die Liste der Befehlszeilenoptionen werden für die CLI angezeigt.

```
vSphere login manager is responsible for vSphere authentication.
It allows vSphere users to securely login and logout to access Harbor images.

Usage:
  docker-credential-vsphere [command]

Available Commands:
  help        Help about any command
  login       Login into specific harbor server and get authentication
  logout      Logout from Harbor server and erase user token

Flags:
  -h, --help   help for docker-credential-vsphere

Use "docker-credential-vsphere [command] --help" for more information about a command.
```

9 Melden Sie sich bei einer Registrierung an.

Überprüfen Sie zunächst die Nutzung:

```
docker-credential-vsphere login -help
Usage:
  docker-credential-vsphere login [harbor-registry] [flags]

Flags:
```

```
  -h, --help               help for login
  -s, --service string     credential store service
      --tlscacert string   location to CA certificate (default "/etc/docker/certs.d/*.crt")
  -u, --user string        vSphere username and password
```

Hinweis Der vSphere Docker Credential Helper erwartet, dass die Benutzerzeichenfolge nur Kleinbuchstaben enthält. Wenn Sie nicht nur Kleinbuchstaben verwenden, funktioniert die Anmeldung möglicherweise, nachfolgende Docker-Befehle jedoch nicht. Stellen Sie sicher, dass Sie nur Kleinbuchstaben für den Benutzernamen verwenden.

Dann melden Sie sich mit dem folgenden Befehl an:

```
docker-credential-vsphere login <container-registry-IP>
```

Das Authentifizierungstoken wird abgerufen und gespeichert, und Sie sind angemeldet.

```
docker-credential-vsphere login 10.179.145.77
Username: administrator@vsphere.local
Password: INFO[0017] Fetched username and password
INFO[0017] Fetched auth token
INFO[0017] Saved auth token
```

10 Melden Sie sich von der Harbor-Registrierung ab.

```
docker-credential-vsphere logout 10.179.145.77
```

Herstellen einer Verbindung zu TKG-Dienst-Clustern mithilfe der vCenter SSO-Authentifizierung

Sie können eine Verbindung zu TKG-Dienst-Clustern mit vCenter Single Sign-On und dem vSphere-Plug-In für kubectl selbst oder in Verbindung mit der Tanzu-CLI herstellen.

Konfigurieren der sicheren Anmeldung für vCenter Single Sign-On-Authentifizierung

Damit Sie sich sicher bei Supervisor und TKG-Dienst-Clustern anmelden können, konfigurieren Sie das vSphere-Plug-In für kubectl mit dem entsprechenden TLS-Zertifikat und achten Sie darauf, dass Sie die neueste Edition des Plug-Ins ausführen.

Supervisor-CA-Zertifikat

Die vSphere IaaS control plane unterstützt vCenter Single Sign-On für den Clusterzugriff mithilfe des vSphere-Plug-In für kubectl-Befehls `kubectl vsphere login …`.

Das vSphere-Plug-In für kubectl verwendet standardmäßig eine sichere Anmeldung und erfordert ein vertrauenswürdiges Zertifikat, wobei standardmäßig das von der vCenter Server-Stammzertifizierungsstelle signierte Zertifikat verwendet wird. Das Plug-In unterstützt das Flag `--insecure-skip-tls-verify` zwar, aber dies wird aus Sicherheitsgründen nicht empfohlen.

Um sich mit dem vSphere-Plug-In für kubectl sicher bei den Supervisorn und TKG-Dienst-Clustern anzumelden, haben Sie zwei Möglichkeiten:

Option	Anleitung
Laden Sie das Zertifikat der vCenter Server-Stammzertifizierungsstelle herunter und installieren Sie es auf jedem Clientcomputer.	Für Linux finden Sie weitere Informationen im folgenden Abschnitt: Herunterladen der vertrauenswürdigen CA-Root-Zertifikate für vCenter und Installieren dieser Zertifikate auf einem Ubuntu-Client Für Windows und Mac finden Sie weitere Informationen im VMware-Knowledgebase-Artikel Vorgehensweise zum Herunterladen und Installieren von vCenter Server-Root-Zertifikaten.
Ersetzen Sie das für Supervisor verwendete VIP-Zertifikat durch ein Zertifikat, das von einer Zertifizierungsstelle signiert wurde, der jede Clientmaschine vertraut.	Weitere Informationen finden Sie unter *Installieren und Konfigurieren der vSphere IaaS-Steuerungsebene*.

Hinweis Weitere Informationen zur vSphere-Authentifizierung, einschließlich vCenter Single Sign-On, Verwaltung und Rotation von vCenter Server-Zertifikaten und Fehlerbehebung, finden Sie in der Dokumentation zur *vSphere-Authentifizierung*.

CA-Zertifikat für TKG-Cluster

Um mithilfe der `kubectl`-CLI eine sichere Verbindung mit dem API-Server des TKG-Clusters herzustellen, müssen Sie das CA-Zertifikat für den TKG-Cluster herunterladen.

Wenn Sie die neueste Edition des vSphere-Plug-In für kubectl verwenden, registriert das Plug-In bei Ihrer ersten Anmeldung beim TKG-Cluster das CA-Zertifikat für den TKG-Cluster in der kubeconfig-Datei. Dieses Zertifikat wird ebenfalls im geheimen Kubernetes-Schlüssel mit dem Namen *`TANZU-KUBERNETES-CLUSTER-NAME`*`-ca` gespeichert. Das Plug-In verwendet das Zertifikat, um die CA-Informationen im Datenspeicher der Zertifizierungsstelle des entsprechenden Clusters aufzufüllen.

Wenn Sie Supervisor aktualisiert haben, stellen Sie sicher, dass das Update auf die neueste Version des Plug-Ins erfolgt.

Herunterladen der vertrauenswürdigen CA-Root-Zertifikate für vCenter und Installieren dieser Zertifikate auf einem Ubuntu-Client

Führen Sie dieses Verfahren aus, um die vertrauenswürdigen CA-Root-Zertifikate für vCenter Server herunterzuladen und auf einem Ubuntu-Client zu installieren, damit Sie sich sicher bei Supervisor- und TKG-Dienst-Clustern mithilfe des vSphere-Plug-In für kubectl anmelden können.

1 Installieren Sie den vSphere-Plug-In für kubectl. Weitere Informationen finden Sie unter Installieren des Kubernetes-CLI-Tools für vSphere.

2 Laden Sie die vertrauenswürdigen CA-Root-Zertifikate für den vCenter Server herunter, auf dem **Arbeitslastverwaltung** aktiviert ist.

```
wget https://VC-IP-or_FQDN/certs/download.zip --no-check-certificate
```

3 Extrahieren Sie den Inhalt der Datei `download.zip` in das aktuelle Verzeichnis.

```
unzip download.zip -d .
```

4 Ändern Sie den Pfad zum Linux-Verzeichnis.

```
cd /certs/lin
```

5 Listen Sie (`ls`) die CA-Zertifikate im Verzeichnis `/certs/lin` auf.

Zwei Zertifikate im PEM-Format sollten angezeigt werden: `*.0` und `*.r1`. Ein PEM-formatiertes Zertifikat ist im Base64-Format lesbar und beginnt mit `----BEGIN CERTIFICATE----`.

6 Hängen Sie die Erweiterung `*.crt` an die Zertifikatsdateien an. Beispiel:

```
cp dbad4059.0 dbad4059.0.crt
```

```
cp dbad4059.r1 dbad4059.r1.crt
```

7 Kopieren Sie die Dateien in das OpenSSL-Zertifikatsverzeichnis in `/etc/ssl/certs`.

```
sudo cp dbad4059.0.crt /etc/ssl/certs
```

```
sudo cp dbad4059.r1.crt /etc/ssl/certs
```

8 Melden Sie sich sicher bei Supervisor an.

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USERNAME
```

9 Melden Sie sich sicher beim TKG-Dienst-Cluster an.

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USERNAME --tanzu-kubernetes-
cluster-name CLUSTER-NAME --tanzu-kubernetes-cluster-namespace VSPHERE-NS
```

Konfigurieren von vSphere-Namespace-Berechtigungen für Benutzer und Gruppen mit vCenter Single Sign-On

Legen Sie die Berechtigungen für den vSphere-Namespace so fest, dass die Benutzer und Gruppen mit vCenter Single Sign-On auf die hier bereitgestellten TKG 2-Cluster zugreifen können.

Nachdem Sie einen vSphere-Namespace erstellt haben, konfigurieren Sie ihn für TKG 2-Cluster, indem Sie Benutzer/Gruppen hinzufügen und Rollen zuweisen. Weitere Informationen finden Sie unter Konfigurieren eines vSphere-Namespace für TKG-Dienst-Cluster.

Voraussetzungen

Benutzer, Gruppen und Rollenberechtigungen werden auf der vSphere-Namespace-Ebene festgelegt. Um auf Supervisor- und TKG 2-Cluster zuzugreifen, müssen Sie zuerst einen vSphere-Namespace erstellen. Weitere Informationen finden Sie unter Erstellen eines vSphere-Namespace für das Hosting von TKG-Dienst-Clustern.

Verfahren

1 Melden Sie sich mithilfe des vSphere Client bei vCenter Server an.

2 Wählen Sie **Arbeitslastverwaltung > Namespaces** aus.

3 Wählen Sie den von Ihnen erstellten vSphere-Namespace aus.

4 Wählen Sie **Berechtigungen > Berechtigungen hinzufügen** aus.

5 **Identitätsquelle**: Wählen Sie **vsphere.local** für vCenter SSO-Benutzer und -Gruppen aus.

Hinweis Wenn Sie einen externen Identitätsanbieter verwenden, finden Sie weitere Informationen unter Herstellen einer Verbindung zu TKG-Clustern auf Supervisor mithilfe eines externen Identitätsanbieters.

6 **Benutzer-/Gruppensuche**: Wählen Sie den vCenter SSO-Benutzer oder die vCenter SSO-Gruppe aus, der bzw. die für TKG-Clustervorgänge oder TKG-Entwickler konfiguriert ist.

7 **Rolle**: Weisen Sie dem Benutzer oder der Gruppe eine Rolle zu, indem Sie die entsprechende Rolle auswählen: **Lesezugriff**, **Schreibzugriff** oder **Besitzer**.

Option	Bezeichnung
Lesezugriff	Kann TKG-Clusterobjekte im vSphere-Namespace lesen. Kubernetes-Rollen sind keine Berechtigungen zugeordnet. Weitere Informationen hierzu finden Sie unter Rollenberechtigungen und -bindungen.
Schreibzugriff	Kann TKG-Clusterobjekte im vSphere-Namespace erstellen, lesen, aktualisieren und löschen. Kann TKG-Cluster betreiben, die im vSphere Namespace als Kubernetes-`cluster-admin` bereitgestellt werden. Weitere Informationen hierzu finden Sie unter Rollenberechtigungen und -bindungen.
Besitzer	Dieselben Berechtigungen wie „Schreibzugriff“, mit der zusätzlichen Berechtigung zum Erstellen und Verwalten von vSphere-Namespaces mithilfe von kubectl. Nur mit vCenter SSO verfügbar. Weitere Informationen hierzu finden Sie unter Rollenberechtigungen und -bindungen.

8 Vervollständigen Sie die Konfiguration des vSphere-Namespace. Weitere Informationen finden Sie unter Konfigurieren eines vSphere-Namespace für TKG-Dienst-Cluster.

Herstellen einer Verbindung zu Supervisor als vCenter Single Sign-On-Benutzer mit Kubectl

Stellen Sie mit dem vSphere-Plug-In für kubectl eine Verbindung zum Supervisor her und authentifizieren Sie sich mit Ihren vCenter Single Sign-On-Anmeldedaten.

Nach der Anmeldung beim Supervisor generiert das vSphere-Plug-In für kubectl den Kontext für den Cluster. In Kubernetes enthält ein Konfigurationskontext einen Cluster, einen Namespace und einen Benutzer. Sie können sich den Clusterkontext in der Datei `.kube/config` ansehen. Diese Datei wird gemeinhin als `kubeconfig`-Datei bezeichnet.

Hinweis Wenn Sie über eine vorhandene `kubeconfig`-Datei verfügen, wird sie an jeden Clusterkontext angehängt. Das vSphere-Plug-In für kubectl berücksichtigt die KUBECONFIG-Umgebungsvariable, die kubectl selbst verwendet. Auch wenn dies nicht erforderlich ist, kann es hilfreich sein, diese Variable vor der Ausführung von `kubectl vsphere login ...` festzulegen, damit die Informationen in eine neue Datei geschrieben werden (und nicht Ihrer aktuellen `kubeconfig`-Datei hinzugefügt werden).

Voraussetzungen

- Installieren des Kubernetes-CLI-Tools für vSphere.
- Rufen Sie Ihre vCenter Single Sign-On-Anmeldedaten ab.
- Bitten Sie den vSphere-Administrator um die IP-Adresse für die Steuerungsebene des Supervisor.
- Beziehen Sie den Namen des vSphere-Namespace von Ihrem vSphere-Administrator.
- Überprüfen Sie, ob Sie über **Bearbeitungsberechtigungen** für den vSphere-Namespace verfügen. Weitere Informationen hierzu finden Sie unter Konfigurieren von vSphere-Namespace-Berechtigungen für Benutzer und Gruppen mit vCenter Single Sign-On.
- Stellen Sie sicher, dass das Zertifikat, das von der Kubernetes-Steuerungsebene bereitgestellt wird, auf Ihrem System als vertrauenswürdig eingestuft wird, indem Sie entweder die signierende Zertifizierungsstelle als vertrauenswürdigen Root installieren oder das Zertifikat direkt als vertrauenswürdigen Root hinzufügen. Weitere Informationen hierzu finden Sie unter Konfigurieren der sicheren Anmeldung für vCenter Single Sign-On-Authentifizierung.

Verfahren

1. Um die Befehlssyntax und die Optionen für die Anmeldung anzuzeigen, führen Sie folgenden Befehl aus:

   ```
   kubectl vsphere login --help
   ```

2. Um eine Verbindung mit dem Supervisor herzustellen, führen Sie den folgenden Befehl aus.

   ```
   kubectl vsphere login --server=<KUBERNETES-CONTROL-PLANE-IP-ADDRESS> --vsphere-username
   <VCENTER-SSO-USER>
   ```

   Sie können sich auch mit einem FQDN anmelden:

   ```
   kubectl vsphere login --server <KUBERNETES-CONTROL-PLANE-FQDN --vsphere-username <VCENTER-
   SSO-USER>
   ```

Beispiel:

```
kubectl vsphere login --server=10.92.42.13 --vsphere-username administrator@example.com
```

```
kubectl vsphere login --server wonderland.acme.com --vsphere-username
administrator@example.com
```

Bei dieser Aktion wird eine Konfigurationsdatei mit dem JSON-Web-Token (JWT) für die Authentifizierung bei der Kubernetes-API erstellt.

3 Geben Sie zur Authentifizierung das Kennwort für den Benutzer ein.

Nachdem Sie eine Verbindung mit dem Supervisor hergestellt haben, werden Ihnen die Konfigurationskontexte angezeigt, auf die zugegriffen werden kann. Beispiel:

```
You have access to the following contexts:
tanzu-ns-1
tkg-cluster-1
tkg-cluster-2
```

4 Führen Sie folgenden `kubectl`-Befehl aus, um Details zu den Konfigurationskontexten anzuzeigen, auf die Sie zugreifen können:

```
kubectl config get-contexts
```

Die CLI zeigt die Details für jeden verfügbaren Kontext an.

5 Verwenden Sie den folgenden Befehl, um zwischen den Kontexten zu wechseln:

```
kubectl config use-context <example-context-name>
```

Herstellen einer Verbindung mit einem TKG-Dienst-Cluster als vCenter Single Sign-On-Benutzer mit Kubectl

Stellen Sie mit dem vSphere-Plug-In für kubectl eine Verbindung zum TKG-Cluster her und authentifizieren Sie sich mit Ihren vCenter Single Sign-On-Anmeldedaten.

Nach der Anmeldung beim Tanzu Kubernetes-Cluster generiert das vSphere-Plug-In für kubectl den Kontext für den Cluster. In Kubernetes enthält ein Konfigurationskontext einen Cluster, einen Namespace und einen Benutzer. Sie können sich den Clusterkontext in der Datei `.kube/config` ansehen. Diese Datei wird gemeinhin als `kubeconfig`-Datei bezeichnet.

Hinweis Wenn Sie über eine vorhandene `kubeconfig`-Datei verfügen, wird sie an jeden Clusterkontext angehängt. Das vSphere-Plug-In für kubectl berücksichtigt die KUBECONFIG-Umgebungsvariable, die kubectl selbst verwendet. Auch wenn dies nicht erforderlich ist, kann es hilfreich sein, diese Variable vor der Ausführung von `kubectl vsphere login ...` festzulegen, damit die Informationen in eine neue Datei geschrieben werden (und nicht Ihrer aktuellen `kubeconfig`-Datei hinzugefügt werden).

Voraussetzungen

Besorgen Sie sich die folgenden Informationen bei Ihrem vSphere-Administrator:

- Rufen Sie Ihre vCenter Single Sign-On-Anmeldedaten ab.
- Rufen Sie die IP-Adresse der Supervisor-Steuerungsebene ab.
- Rufen Sie den Namen des vSphere-Namespace ab.
- Installieren des Kubernetes-CLI-Tools für vSphere.

Verfahren

1 Um die Befehlssyntax und die Optionen für die Anmeldung anzuzeigen, führen Sie folgenden Befehl aus:

```
kubectl vsphere login --help
```

2 Um eine Verbindung mit dem Tanzu Kubernetes-Cluster herzustellen, führen Sie den folgenden Befehl aus.

```
kubectl vsphere login --server=SUPERVISOR-CLUSTER-CONTROL-PLANE-IP-OR-FQDN
--tanzu-kubernetes-cluster-name TKG-CLUSTER-NAME
--tanzu-kubernetes-cluster-namespace VSPHERE-NAMESPACE
--vsphere-username VCENTER-SSO-USER-NAME
```

Beispiel:

```
kubectl vsphere login --server=10.92.42.137
--tanzu-kubernetes-cluster-name tkg-cluster-01
--tanzu-kubernetes-cluster-namespace tkg-cluster-ns
--vsphere-username operator@example.com
```

Oder, wenn Supervisor mit einem vollqualifizierten Domänennamen (FQDN) aktiviert wurde:

```
kubectl vsphere login --server=wonderland.acme.com
--tanzu-kubernetes-cluster-name tkg-cluster-01
--tanzu-kubernetes-cluster-namespace tkg-cluster-ns
--vsphere-username operator@example.com
```

Bei dieser Aktion wird eine Konfigurationsdatei mit dem JSON-Web-Token (JWT) für die Authentifizierung bei der Kubernetes-API erstellt.

3 Geben Sie Ihr vCenter Single Sign-On-Kennwort ein, um sich zu authentifizieren.

Wenn der Vorgang erfolgreich ist, wird die Meldung `Logged in successfully` angezeigt, und Sie können kubectl-Befehle für den Cluster ausführen. Wenn der Befehl `Error from server (Forbidden)` zurückgibt, bedeutet dieser Fehler in der Regel, dass Sie nicht über die erforderlichen Berechtigungen verfügen.

4 Führen Sie zum Abrufen einer Liste mit für Sie verfügbaren Kontexten folgenden Befehl aus:

```
kubectl config get-contexts
```

Dieser Befehl listet die Konfigurationskontexte auf, auf die Sie zugreifen können. Ein Konfigurationskontext für den Zielcluster, wie z. B. `tkg-cluster-01`, wird angezeigt.

5 Zur Verwendung des Kontexts für den Zielcluster führen Sie folgenden Befehl aus:

```
kubectl config use-context CLUSTER-NAME
```

6 Führen Sie zum Auflisten von Clusterknoten folgenden Befehl aus:

```
kubectl get nodes
```

Die Steuerungsebene und die Worker-Knoten in diesem Cluster werden angezeigt.

7 Führen Sie zum Auflisten aller Cluster-Pods folgenden Befehl aus:

```
kubectl get pods -A
```

Sie sehen alle Pods in diesem Cluster über alle Kubernetes-Namespaces hinweg, auf die Sie Zugriff haben. Wenn Sie keine Arbeitslasten bereitgestellt haben, sehen Sie im Standard-Namespace keine Pods.

Gewähren von vCenter SSO-Zugriff auf TKG-Dienst-Cluster für Entwickler

Entwicklerbenutzer und Entwicklungsgruppen sind die Zielbenutzer von TKG-Dienst-Clustern. Sobald ein TKG-Dienst-Cluster bereitgestellt wurde, können Sie Entwicklerzugriff per vCenter Single Sign-On-Authentifizierung oder über einen unterstützten externen Identitätsanbieter gewähren.

Authentifizierung für Entwickler

Ein Clusteradministrator kann anderen Benutzern, z. B. Entwicklern, Clusterzugriff gewähren. Entwickler können Pods für Cluster direkt über ihre Benutzerkonten oder indirekt über Dienstkonten bereitstellen.

- Für die Authentifizierung über Benutzerkonten bieten TKG-Dienst-Cluster Unterstützung für vCenter Single Sign-On-Benutzer und -Gruppen. Der Benutzer oder die Gruppe kann sich lokal in vCenter Server befinden oder von einem unterstützten Verzeichnisserver aus synchronisiert werden.
- Externe OIDC-Benutzer und -Gruppen werden vSphere-Namespace-Rollen direkt zugeordnet.
- Für die Authentifizierung über Dienstkonten können Sie Diensttoken verwenden. Weitere Informationen dazu finden Sie in der Kubernetes-Dokumentation.

Hinzufügen von Entwicklern zu einem Cluster

So gewähren Sie Entwicklern Clusterzugriff:

1 Definieren Sie ein Role- oder ClusterRole-Objekt für den Benutzer bzw. die Gruppe und wenden Sie es auf den Cluster an. Weitere Informationen dazu finden Sie in der Kubernetes-Dokumentation.

2 Erstellen Sie ein RoleBinding- oder ClusterRoleBinding-Objekt für den Benutzer bzw. die Gruppe und wenden Sie es auf den Cluster an. Betrachten Sie das folgende Beispiel.

RoleBinding – Beispiel

Um einem vCenter Single Sign-On-Benutzer oder einer vCenter Single Sign-On-Gruppe Zugriff zu gewähren, muss im RoleBinding-Objekt unter „subjects“ einer der folgenden Werte für den Parameter `name` angegeben sein.

Tabelle 4-2. Unterstützte Felder für Benutzer und Gruppen

Bereich	Beschreibung
`sso:USER-NAME@DOMAIN`	Beispielsweise ein lokaler Benutzername wie `sso:joe@vsphere.local`.
`sso:GROUP-NAME@DOMAIN`	Beispielsweise ein von einem in vCenter Server integrierten Verzeichnisserver stammender Gruppenname wie `sso:devs@ldap.example.com`.

Im folgenden Beispiel für ein RoleBinding-Objekt wird der lokale vCenter Single Sign-On-Benutzer mit dem Namen „Joe“ an das standardmäßige ClusterRole-Objekt mit dem Namen `edit` gebunden. Diese Rolle ermöglicht Lese-/Schreibzugriff auf die meisten Objekte in einem Namespace. In diesem Fall ist dies der Namespace `default`.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rolebinding-cluster-user-joe
  namespace: default
roleRef:
  kind: ClusterRole
  name: edit                              #Default ClusterRole
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: User
  name: sso:joe@vsphere.local             #sso:<username>@<domain>
  apiGroup: rbac.authorization.k8s.io
```

Herstellen einer Verbindung zu Supervisor mithilfe der Tanzu-CLI und der vCenter SSO-Authentifizierung

Führen Sie die folgenden Schritte aus, um mithilfe der Tanzu-CLI eine Verbindung zu Supervisor herzustellen und sich als vCenter Single Sign-On-Benutzer zu authentifizieren.

Voraussetzungen

Erfüllen Sie die folgenden Voraussetzungen.

1 Installieren und konfigurieren Sie Kubernetes-CLI-Tools für vSphere. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu TKG-Dienst-Clustern mithilfe der vCenter SSO-Authentifizierung.

2 Installieren und initialisieren Sie die Tanzu-CLI. Weitere Informationen finden Sie unter Installieren der Tanzu-CLI zur Verwendung mit TKG-Dienst-Clustern.

Herstellen einer Verbindung zu Supervisor mit Tanzu-CLI und vCenter SSO

Führen Sie die folgenden Schritte aus.

1 Stellen Sie als vCenter SSO-Benutzer eine Verbindung mit Supervisor her.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS --vsphere-username
VCENTER-SSO-USER
```

Mit dieser Aktion wird kubeconfig mit dem Supervisor-Kontext aufgefüllt, den die Tanzu-CLI verwendet.

2 Wechseln Sie zum vSphere-Namespace-Kontext für Supervisor.

```
kubectl config get-contexts
```

```
kubectl config use-context <SUPERVISOR-CONTROL-PLANE-IP-ADDRESS>
```

3 Melden Sie sich bei Supervisor mit der Tanzu-CLI und vCenter SSO an.

```
tanzu context create context_name --kubeconfig ~/.kube/config --kubecontext SUPERVISOR-
CONTROL-PLANE-IP-ADDRESS
```

Dabei gilt:

- `context_name` ist der benutzerdefinierte Name für diesen Kontext, z. B. „supervisor“.
- `--kubeconfig ~/.kube/config` ist der Pfad zur lokalen `kubeconfig`-Datei, der standardmäßig `~/.kube/config` lautet und von der Umgebungsvariable `KUBECONFIG` festgelegt wird, die den Supervisor-Konfigurationskontext für Ihren vCenter SSO-Benutzer enthält.
- `--kubecontext SUPERVISOR-CONTROL-PLANE-IP-ADDRESS` ist der Kontext von Supervisor, der mit `SUPERVISOR_IP` identisch ist, z. B. `10.179.144.55`.

4 Führen Sie Tanzu-CLI-Befehle aus und überprüfen Sie die Konnektivität.

```
tanzu plugin list
```

```
tanzu cluster list -n VSPHERE-NS-FOR-TKG
```

Herstellen einer Verbindung zu TKG-Clustern auf Supervisor mithilfe eines externen Identitätsanbieters

Sie können eine Verbindung zu TKG-Clustern auf Supervisor mithilfe eines externen Identitätsanbieters und des Pinniped-Authentifizierungsdiensts für Kubernetes herstellen.

Konfigurieren eines externen IDP für die Verwendung mit TKG-Dienstclustern

Sie können Supervisor mit jedem OIDC-konformen Identitätsanbieter (IDP), z. B. Okta, konfigurieren. Um die Integration abzuschließen, konfigurieren Sie den IDP mit der Callback-URL für Supervisor.

Unterstützte externe OIDC-Anbieter

Sie können Supervisor mit jedem OIDC-konformen Identitätsanbieter konfigurieren. Die Tabelle enthält gängige Beispiele und Links zu Konfigurationsanweisungen.

Externer IDP	Konfiguration
Okta	Beispiel für eine OIDC-Konfiguration mit Okta Weitere Informationen finden Sie unter Konfigurieren von Okta als OIDC-Anbieter für Pinniped
Workspace ONE	Konfigurieren von Workspace ONE Access als OIDC-Anbieter für Pinniped
Dex	Konfigurieren von Dex als OIDC-Anbieter für Pinniped
GitLab	Konfigurieren von GitLab als OIDC-Anbieter für Pinniped
Google OAuth	Verwenden von Google OAuth 2

Konfigurieren des IDP mit der Callback-URL für Supervisor

Supervisor fungiert als OAuth 2.0-Client für den externen Identitätsanbieter. Die Supervisor-Callback-URL ist die Umleitungs-URL, die zum Konfigurieren des externen Identitätsanbieters verwendet wird. Die Callback-URL hat das Format *https://SUPERVISOR-VIP/wcp/pinniped/callback*.

Hinweis Bei der IDP-Registrierung wird die Callback-URL in dem OIDC-Anbieter, den Sie konfigurieren, möglicherweise als „Weiterleitungs-URL“ bezeichnet.

Wenn Sie den externen Identitätsanbieter für die Verwendung mit TKG auf Supervisor konfigurieren, übermitteln Sie dem externen Identitätsanbieter die **Callback-URL**, die in vCenter Server im Bildschirm **Arbeitslastverwaltung > Supervisoren > Konfigurieren > Identitätsanbieter** verfügbar ist.

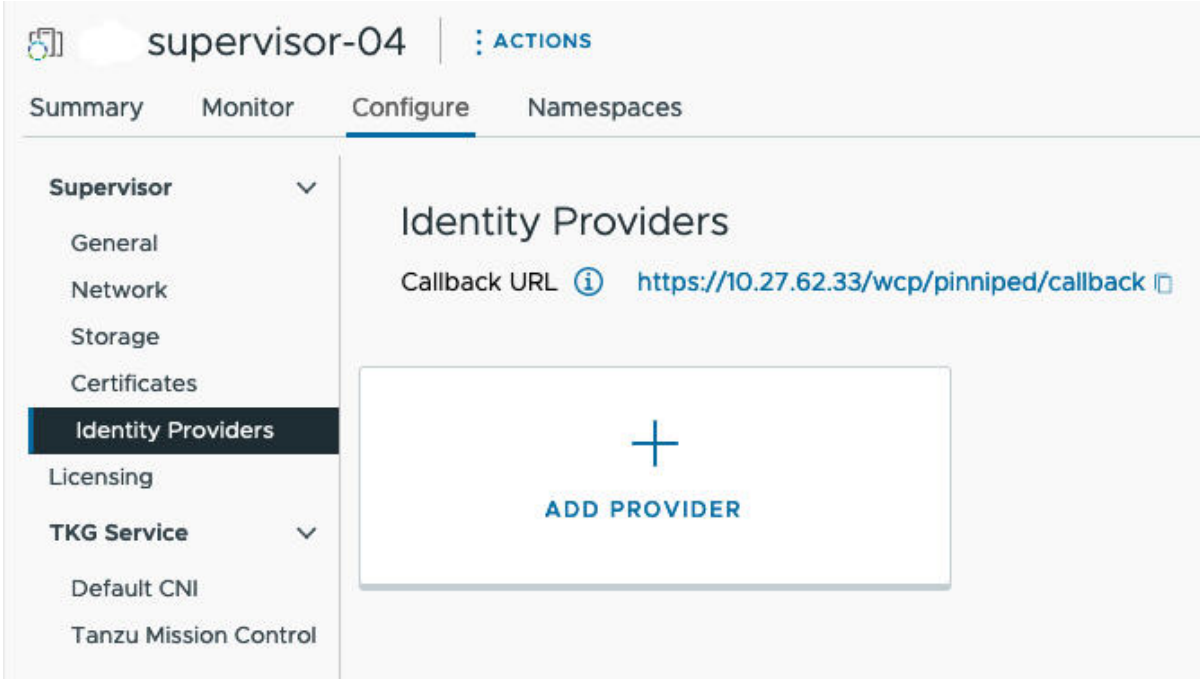
Beispiel für eine OIDC-Konfiguration mit Okta

Mit Okta können sich Benutzer mit dem OpenID Connect-Protokoll bei Anwendungen anmelden. Wenn Sie Okta als externen Identitätsanbieter für Tanzu Kubernetes Grid auf Supervisor konfigurieren, steuern die Pinniped-Pods auf Supervisor und auf Tanzu Kubernetes Grid-Clustern den Benutzerzugriff für vSphere-Namespaces und für Arbeitslastcluster.

1 Kopieren Sie die Callback-URL des Identitätsanbieters, die Sie zum Erstellen einer OIDC-Verbindung zwischen Okta und vCenter Server benötigen.

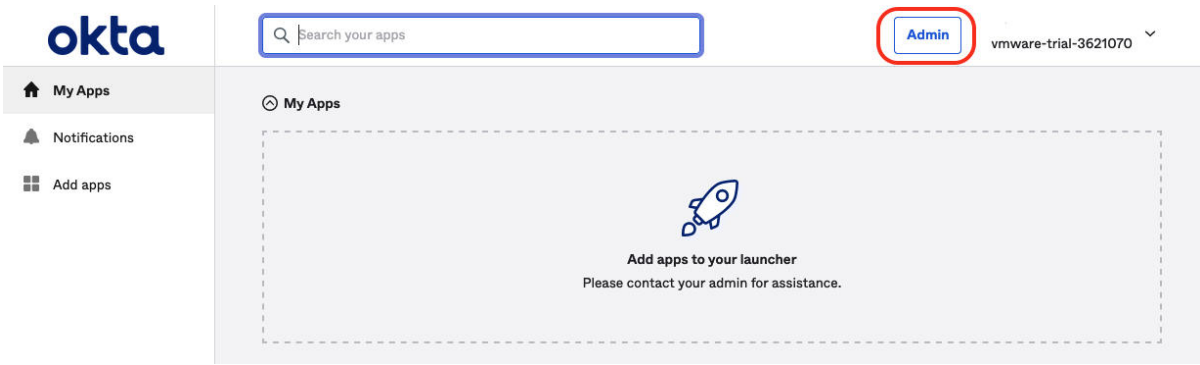
Rufen Sie die Callback-URL des Identitätsanbieters im vSphere Client unter **Arbeitslastverwaltung > Supervisoren > Konfigurieren > Identitätsanbieter** ab. Kopieren Sie diese URL an einen temporären Speicherort.

Abbildung 4-2. IDP-Callback-URL



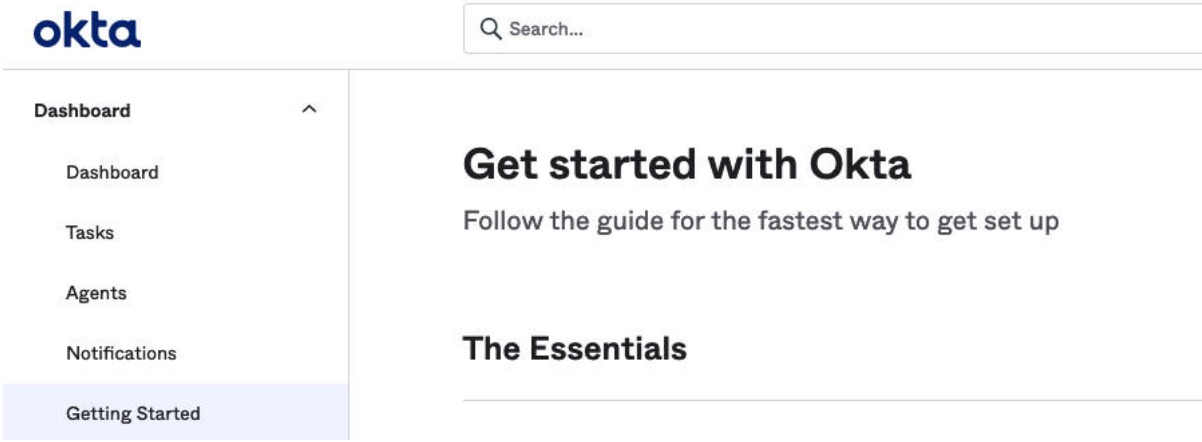
2 Melden Sie sich beim Okta-Konto für Ihre Organisation an oder erstellen Sie ein Testkonto unter https://www.okta.com/. Klicken Sie auf die Schaltfläche **Admin**, um die Okta-Verwaltungskonsole zu öffnen.

Abbildung 4-3. Okta-Verwaltungskonsole



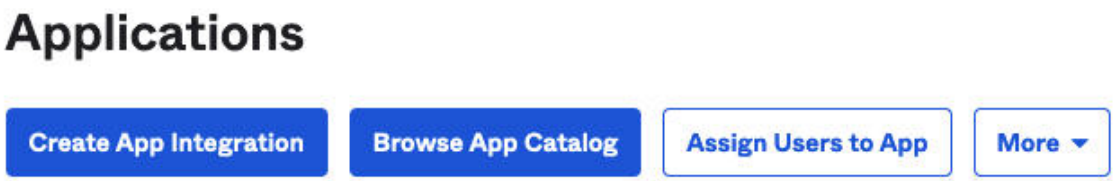
3 Navigieren Sie in der Verwaltungskonsole auf der Seite „Erste Schritte“ zu **Anwendungen > Anwendungen**.

Abbildung 4-4. Okta – Erste Schritte



4 Wählen Sie die Option **App-Integration erstellen** aus.

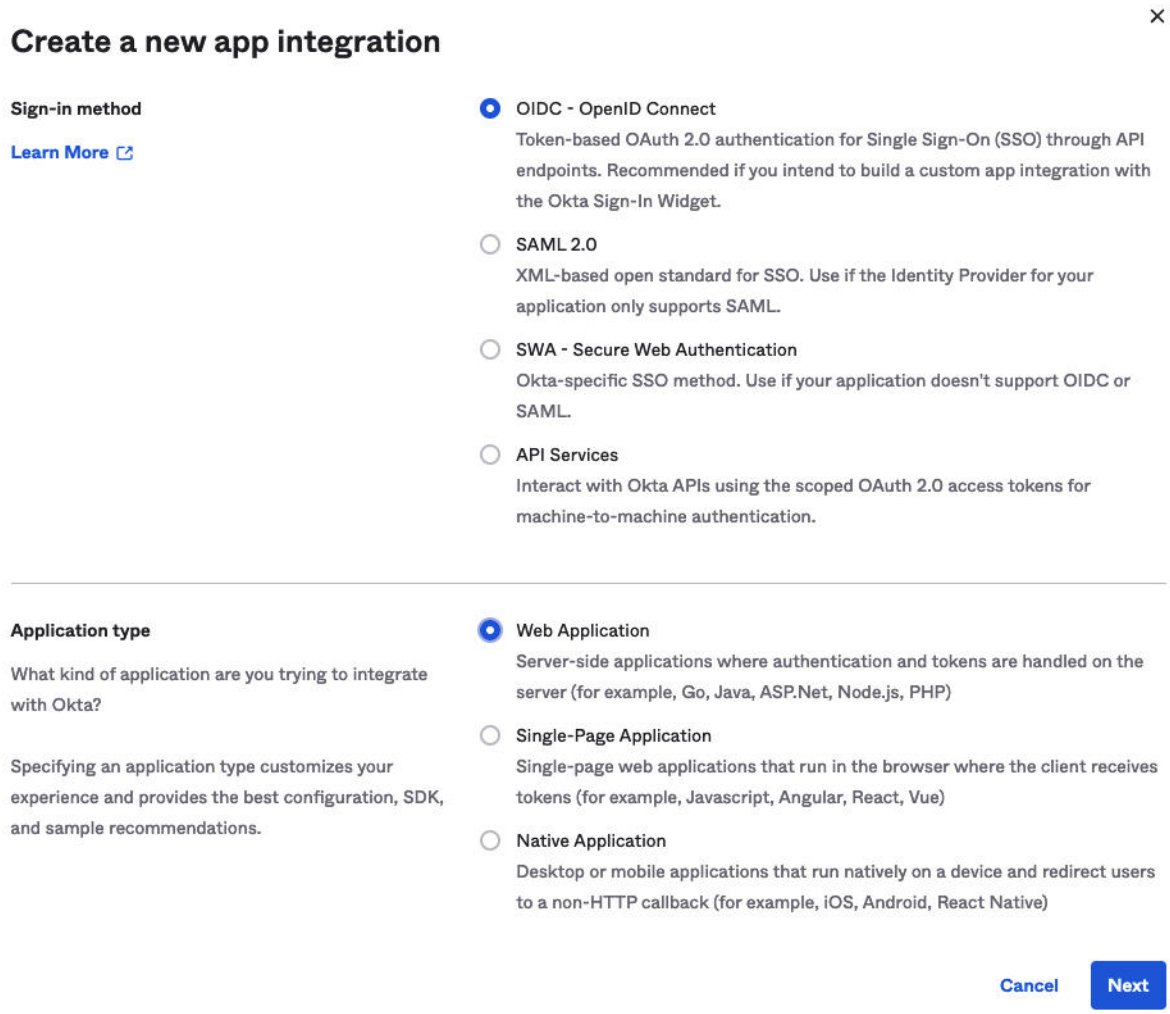
Abbildung 4-5. Okta – App-Integration erstellen



5 Erstellen Sie die neue App-Integration.

- Legen Sie die Anmeldemethode auf **OIDC - OpenID Connect** fest.
- Legen Sie den Anwendungstyp auf **Webanwendung** fest.

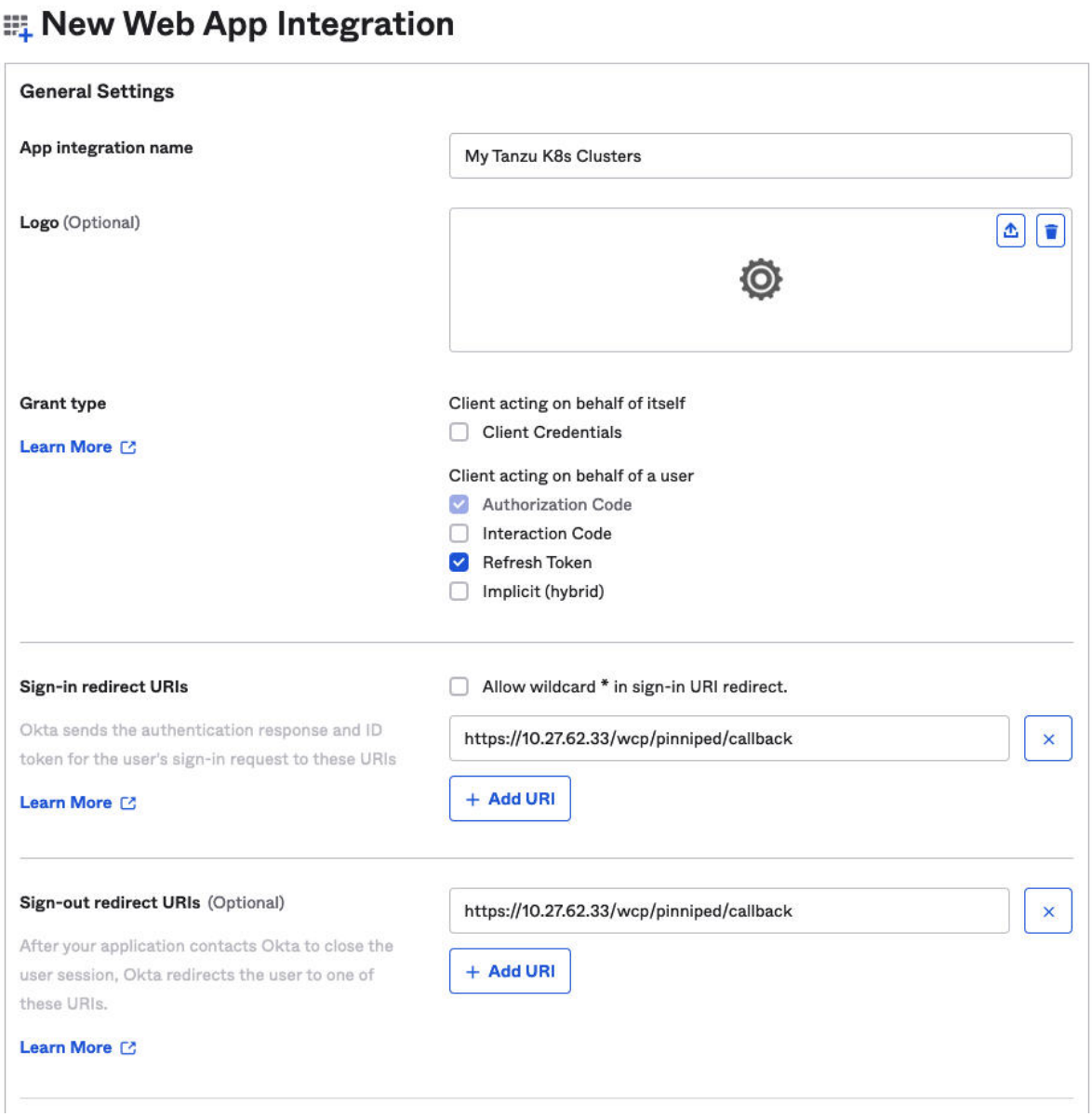
Abbildung 4-6. Okta-Anmeldemethode und Anwendungstyp



6 Konfigurieren Sie die Details der Okta-Webanwendungsintegration.

- Geben Sie einen **Namen für die App-Integration** an, der eine benutzerdefinierte Zeichenfolge ist.
- Geben Sie den **Gewährungstyp** an: Wählen Sie **Autorisierungscode** und außerdem **Token aktualisieren** aus.
- Weiterleitungs-URIs für die Anmeldung: Geben Sie die Callback-URL des Identitätsanbieters ein, die Sie von Supervisor kopiert haben (siehe Schritt 1), z. B. https://10.27.62.33/wcp/pinnipend/callback.
- Weiterleitungs-URIs für die Abmeldung: Geben Sie die Callback-URL des Identitätsanbieters ein, die Sie von Supervisor kopiert haben (siehe Schritt 1), z. B. https://10.27.62.33/wcp/pinnipend/callback.

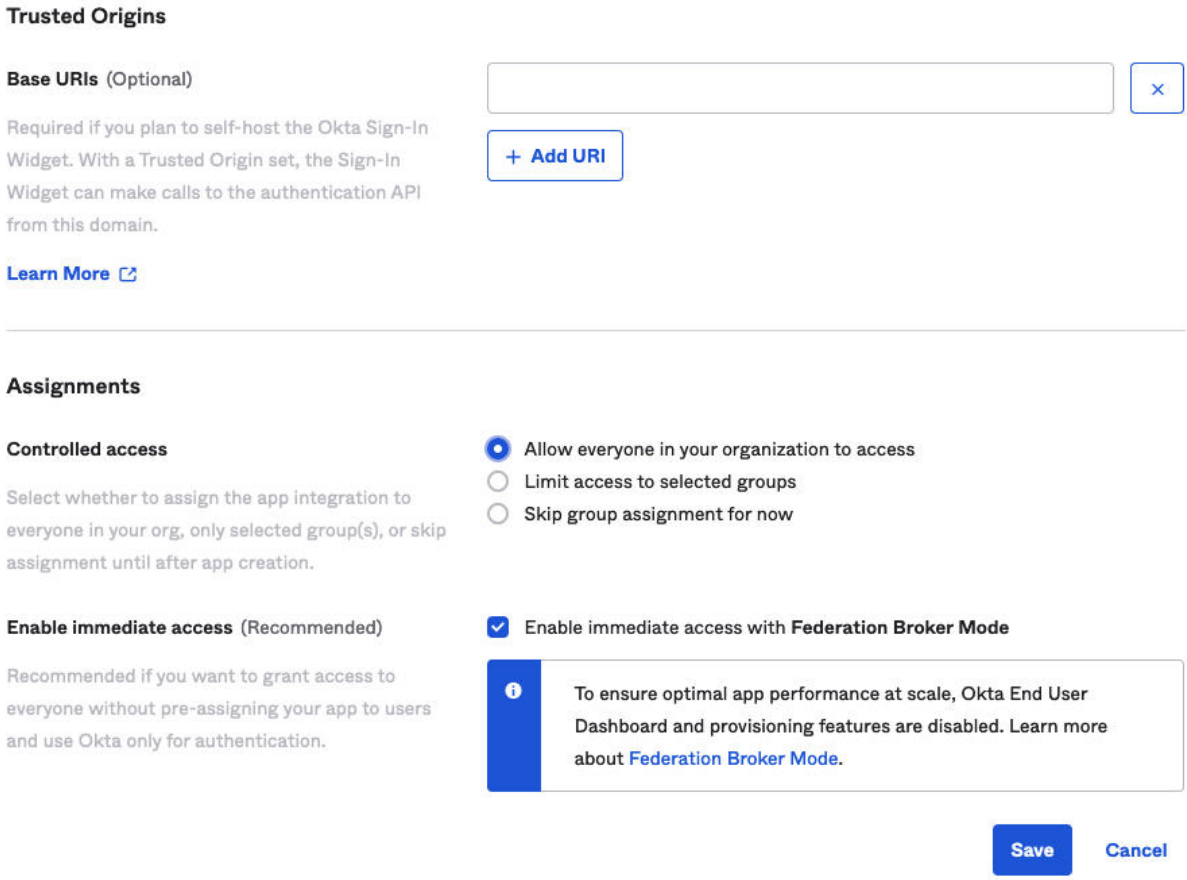
Abbildung 4-7. Details der Okta-Webanwendungsintegration



7 Konfigurieren Sie die Benutzerzugriffssteuerung.

Im Abschnitt **Zuweisungen > Kontrollierter Zugriff** können Sie optional steuern, welche der in Ihrer Organisation vorhandenen Okta-Benutzer auf Tanzu Kubernetes Grid-Cluster zugreifen können. Im Beispiel gewähren Sie allen in der Organisation definierten Benutzern Zugriff.

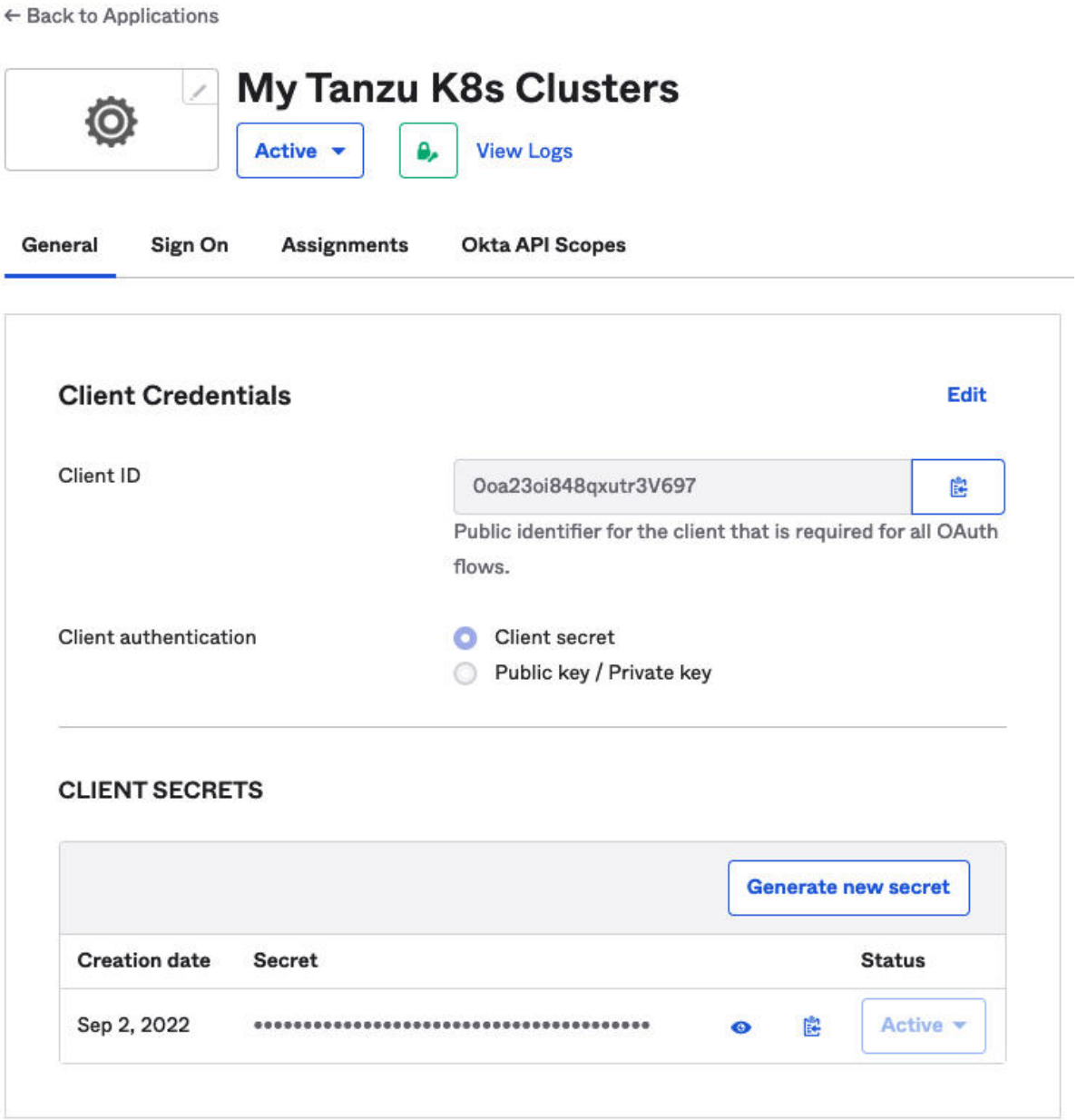
Abbildung 4-8. Okta-Zugriffssteuerung



8 Klicken Sie auf **Speichern** und kopieren Sie die **Client-ID** und den **geheimen Clientschlüssel**, die zurückgegeben werden.

Wenn Sie die OKTA-Konfiguration speichern, stellt Ihnen die Verwaltungskonsole eine **Client-ID** und einen **geheimen Clientschlüssel** zur Verfügung. Kopieren Sie beide Elemente, da Sie diese benötigen, um Supervisor mit einem externen Identitätsanbieter zu konfigurieren.

Abbildung 4-9. OIDC-Client-ID und geheimer Schlüssel

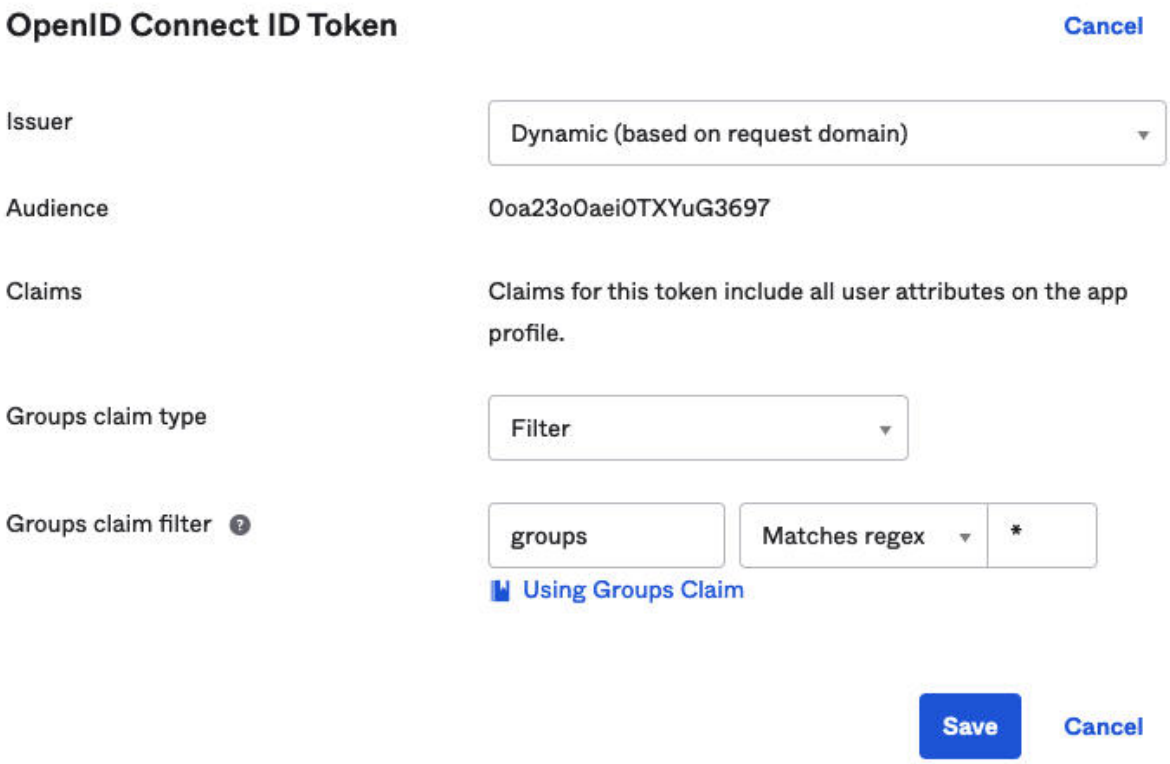


9 Konfigurieren Sie das OpenID Connect-ID-Token.

Klicken Sie auf die Registerkarte **Anmelden**. Klicken Sie im Abschnitt **OpenID Connect ID-Token** auf den Bearbeitungslink, geben Sie als **Gruppenanspruchstyp** „Filter“ ein und **speichern** Sie die Einstellungen.

Wenn der Anspruchname „Gruppen“ allen Gruppen entsprechen soll, wählen Sie **Gruppen > Entspricht regex >** * aus.

Abbildung 4-10. OpenID Connect-ID-Token

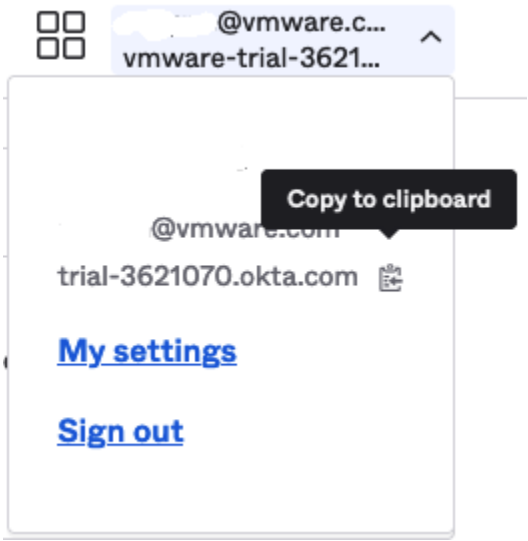


10 Kopieren Sie die **Aussteller-URL**.

Um Supervisor zu konfigurieren, benötigen Sie die **Aussteller-URL** sowie die **Client-ID** und den **geheimen Clientschlüssel**.

Kopieren Sie die **Aussteller-URL** aus der Okta-Verwaltungskonsole.

Abbildung 4-11. Okta-Aussteller-URL



Registrieren eines externen IDP bei Supervisor

Um mithilfe der Tanzu-CLI eine Verbindung zu Tanzu Kubernetes Grid 2.0-Clustern auf Supervisor herzustellen, registrieren Sie Ihren OIDC-Anbieter bei Supervisor.

Voraussetzungen

Bevor Sie einen externen ODIC-Anbieter bei Supervisor registrieren, müssen Sie die folgenden Voraussetzungen erfüllen:

- Aktivieren Sie die Arbeitslastverwaltung und stellen Sie eine Supervisor-Instanz bereit. Weitere Informationen finden Sie unter Ausführen von TKG 2.0-Clustern auf Supervisor.
- Konfigurieren Sie einen externen OpenID Connect-Identitätsanbieter mit der Supervisor-Callback-URL. Weitere Informationen finden Sie unter Konfigurieren eines externen IDP für die Verwendung mit TKG-Dienstclustern.
- Rufen Sie die Client-ID, den geheimen Clientschlüssel und die Aussteller-URL des externen IDP ab. Weitere Informationen finden Sie unter Konfigurieren eines externen IDP für die Verwendung mit TKG-Dienstclustern.

Registrieren eines externen IDP bei Supervisor

Supervisor führt die Pinniped-Supervisor- und Pinniped-Concierge-Komponenten als Pods aus. Tanzu Kubernetes Grid-Cluster führen nur die Pinniped-Concierge-Komponente als Pods aus. Weitere Informationen zu diesen Komponenten und deren Interaktion finden Sie in der Dokumentation zum Pinniped-Authentifizierungsdienst.

Sobald Sie einen externen Identitätsanbieter bei Supervisor registriert haben, aktualisiert das System die Pinniped-Supervisor- und Pinniped-Concierge-Pods auf Supervisor und die Pinniped-Concierge-Pods in Tanzu Kubernetes Grid-Clustern. Alle in dieser Supervisor-Instanz ausgeführten Tanzu Kubernetes Grid-Cluster werden automatisch mit demselben externen Identitätsanbieter konfiguriert.

Führen Sie das folgende Verfahren aus, um einen externen ODIC-Anbieter bei Supervisor zu registrieren:

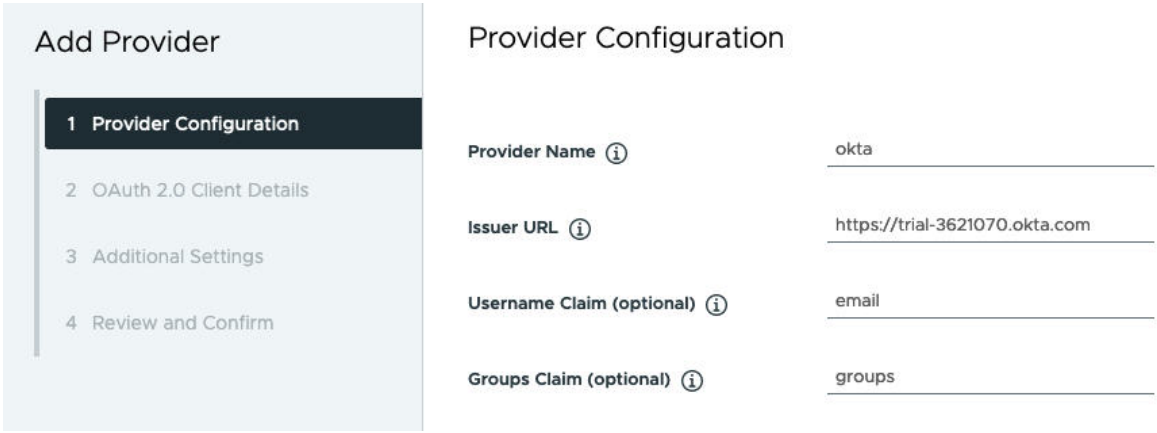
1 Melden Sie sich über vSphere Client bei vCenter Server an.

2 Wählen Sie **Arbeitslastverwaltung > Supervisoren > Konfigurieren > Identitätsanbieter** aus.

3 Klicken Sie auf das Pluszeichen, um den Registrierungsvorgang zu starten.

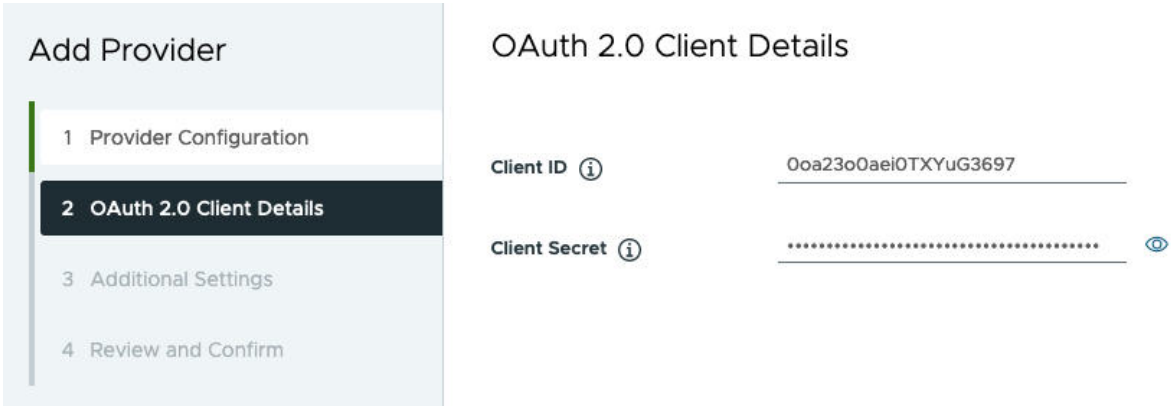
4 Konfigurieren Sie den Identitätsanbieter. Weitere Informationen finden Sie unter Konfiguration des OIDC-Anbieters.

Abbildung 4-12. Konfiguration des OIDC-Anbieters



5 Konfigurieren Sie OAuth 2.0-Client-Details. Weitere Informationen finden Sie unter OAuth 2.0-Client-Details.

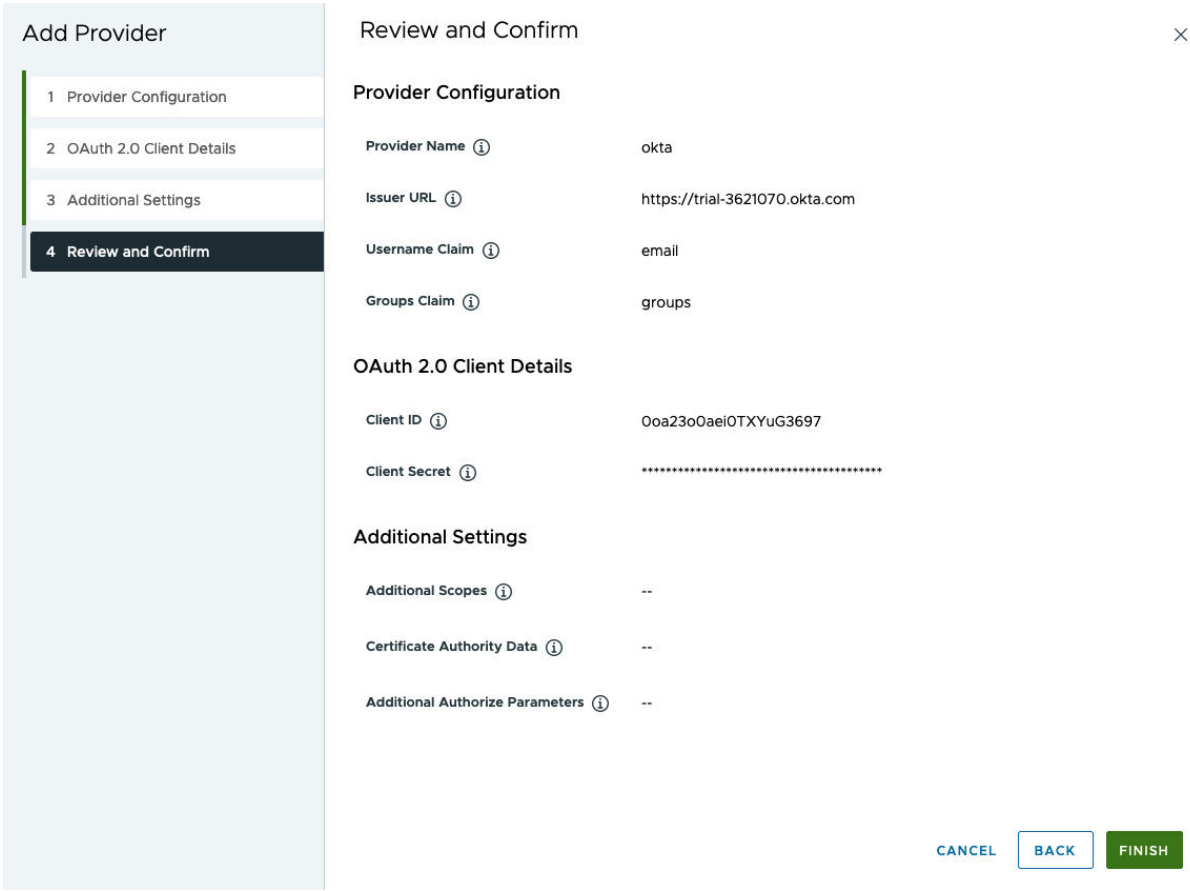
Abbildung 4-13. OAuth 2.0-Client-Details



6 Konfigurieren Sie zusätzliche Einstellungen. Weitere Informationen finden Sie unter Weitere Einstellungen.

7 Bestätigen Sie die Anbietereinstellungen.

Abbildung 4-14. Anbietereinstellungen bestätigen



8 Klicken Sie auf **Beenden**, um die OIDC-Anbieterregistrierung abzuschließen.

Konfiguration des OIDC-Anbieters

Beachten Sie die folgenden Konfigurationsdetails des Anbieters beim Registrieren eines externen OIDC-Anbieters bei Supervisor.

Tabelle 4-3. Konfiguration des OIDC-Anbieters

Bereich	Gewichtung	Beschreibung
Anbietername	Erforderlich	Benutzerdefinierter Name für den externen Identitätsanbieter.
Aussteller-URL	Erforderlich	Die URL zum Identitätsanbieter, der Token ausstellt. Die OIDC-Erkennungs-URL wird von der Aussteller-URL abgeleitet. Für Okta sieht die Aussteller-URL beispielsweise wie folgt aus und kann über die Admin-Konsole abgerufen werden: *https://trial-4359939-admin.okta.com*.

Tabelle 4-3. Konfiguration des OIDC-Anbieters (Fortsetzung)

Bereich	Gewichtung	Beschreibung
Benutzernamensanspruch	Optional	Der Anspruch vom ID-Token des Upstream-Identitätsanbieters oder vom Benutzerinformations-Endpoint, der überprüft werden soll, um den Benutzernamen für den angegebenen Benutzer abzurufen. Wenn Sie dieses Feld leer lassen, wird die Upstream-Aussteller-URL mit dem Anspruch „sub“ verkettet, um den Benutzernamen zu generieren, der mit Kubernetes verwendet werden soll. Dieses Feld gibt an, was Pinniped vom Upstream-ID-Token aus suchen soll, um die Authentifizierung zu ermitteln. Wenn keine Angaben gemacht werden, wird die Benutzeridentität als *https://IDP-ISSUER?sub=UUID* formatiert.
Gruppenanspruch	Optional	Der Anspruch vom ID-Token des Upstream-Identitätsanbieters oder vom Benutzerinformations-Endpoint, der überprüft werden soll, um die Gruppen für den angegebenen Benutzer abzurufen. Wenn Sie dieses Feld leer lassen, werden keine Gruppen des Upstream-Identitätsanbieters verwendet. Das Feld „Gruppenanspruch“ teilt Pinniped mit, was aus dem Upstream-ID-Token zur Authentifizierung der Benutzeridentität zu entnehmen ist.

OAuth 2.0-Client-Details

Beachten Sie die folgenden OAuth 2.0-Client-Details des Anbieters, wenn Sie einen externen OIDC-Anbieter bei Supervisor registrieren.

Tabelle 4-4. OAuth 2.0-Client-Details

OAuth 2.0-Client-Details	Gewichtung	Beschreibung
Client-ID	Erforderlich	Client-ID des externen IDP
Geheimer Clientschlüssel	Erforderlich	Geheimer Clientschlüssel vom externen IDP

Weitere Einstellungen

Beachten Sie die folgenden zusätzlichen Einstellungen, wenn Sie einen externen OIDC-Anbieter bei Supervisor registrieren.

Tabelle 4-5. Weitere Einstellungen

Einstellung	Gewichtung	Beschreibung
Zusätzliche Geltungsbereiche	Optional	Zusätzliche Geltungsbereiche, die in Token angefordert werden müssen
Daten zur Zertifizierungsstelle	Optional	Daten der TLS-Zertifizierungsstelle für eine sichere externe IDP-Verbindung
Zusätzliche Autorisierungsparameter	Optional	Zusätzliche Parameter während der OAuth2-Autorisierungsanforderung

Konfigurieren von vSphere-Namespace-Berechtigungen für Benutzer und Gruppen externer Identitätsanbieter

Um den TKG 2.0-Clusterzugriff für OIDC-Benutzer zu konfigurieren, konfigurieren Sie den vSphere-Namespace mit Rollenberechtigungen für Benutzer und Gruppen externer Identitätsanbieter.

Konfigurieren von vSphere-Namespace-Berechtigungen für Benutzer und Gruppen externer Identitätsanbieter

Ein TKG 2.0-Cluster auf Supervisor wird in einem vSphere-Namespace bereitgestellt. Nachdem Sie einen externen OIDC-Anbieter bei Supervisor registriert haben, konfigurieren Sie den vSphere-Namespace mit Rollenberechtigungen für Benutzer und Gruppen externer OIDC-Anbieter. Mit diesem Vorgang werden die Rollenbindungen für den externen OIDC-Anbieter in jedem TKG 2.0-Cluster in diesem vSphere-Namespace erstellt. Wenn der vSphere-Namespace bereits vorhanden ist, werden die Rollenbindungen aktualisiert.

Hinweis Wenn Sie einen externen IDP auf einem Supervisor registriert haben, werden alle TKG 2.0-Cluster, die auf diesem Supervisor erstellt wurden, automatisch mit dem externen IDP über die Pinniped-Komponenten konfiguriert.

1. Registrieren Sie einen externen Identitätsanbieter bei Supervisor.

   Weitere Informationen finden Sie unter Registrieren eines externen IDP bei Supervisor.

2. Erstellen Sie einen vSphere-Namespace für einen oder mehrere TKG-Cluster oder wählen Sie einen vorhandenen vSphere-Namespace aus.

   Weitere Informationen finden Sie unter Erstellen eines vSphere-Namespace für das Hosting von TKG-Dienst-Clustern.

3. Konfigurieren Sie Benutzer und Rollen für den vSphere-Namespace.

Wählen Sie den externen OIDC-Anbieter als Identitätsquelle aus, fügen Sie Benutzer hinzu und weisen Sie Rollen zu.

a Aktivieren Sie den vSphere-Namespace.

b Wählen Sie **Berechtigungen > Berechtigungen hinzufügen** aus.

c **Identitätsquelle**: Wählen Sie den bei Supervisor registrierten externen Identitätsanbieter aus.

Der **Anbietername**, den Sie zur Registrierung des externen IDP verwendet haben, sollte im Dropdown-Menü angezeigt werden. Ist dies nicht der Fall, überprüfen Sie die Konfiguration.

d **Benutzer/Gruppensuche**: Geben Sie den Benutzer- oder Gruppennamen ein. Die Texteingabe erfolgt als Freiform-Zeichenfolge.

Benutzer und Gruppen von einem externen Identitätsanbieter werden nicht mit vCenter Server synchronisiert und können nicht ausgewählt werden. Sie müssen den Zeichenfolgenwert eingeben, in der Regel eine E-Mail-Adresse. Es ist kein Präfix vorhanden, daher können Sie z. B. „jdoe@acme.com“ eingeben.

e **Rolle**: Weisen Sie den Benutzer oder die Gruppe einer Rolle zu, indem Sie die Rolle auswählen, entweder **Lesezugriff** oder **Schreibzugriff**.

Hinweis Die Rolle „Besitzer“ ist nicht für die Verwendung mit einem externen Identitätsanbieter verfügbar.

4 Vervollständigen Sie die Konfiguration des vSphere-Namespace.

Weitere Informationen finden Sie unter Konfigurieren eines vSphere-Namespace für TKG-Dienst-Cluster.

Herstellen einer Verbindung zu Supervisor mit der Tanzu-CLI und einem externen IDP

Führen Sie die folgenden Schritte aus, um mithilfe der Tanzu-CLI eine Verbindung zu Supervisor herzustellen.

Voraussetzungen

Erfüllen Sie die folgenden Voraussetzungen.

1 Registrieren Sie eine externen Identitätsanbieter, der OIDC-konform ist, bei Supervisor. Weitere Informationen finden Sie unter Registrieren eines externen IDP bei Supervisor.

2 Gewähren Sie OIDC-Benutzern und -Gruppen Zugriff auf einen vSphere-Namespace. Weitere Informationen finden Sie unter Konfigurieren von vSphere-Namespace-Berechtigungen für Benutzer und Gruppen externer Identitätsanbieter.

Herstellen einer Verbindung mit Supervisor mithilfe der Tanzu-CLI

Führen Sie die folgenden Schritte aus.

Hinweis Wenn Sie TKG-Dienst 3.1 oder höher verwenden, laden Sie das CLI-Plug-In pinniped-auth mit derselben Version des TKG-Dienstes herunter. Laden Sie außerdem das imgpkg cli-Plug-In mit der neuesten Version herunter. Ausführliche Informationen finden Sie in der Produktdokumentation zur Tanzu-CLI.

1 Installieren und initialisieren Sie die Tanzu-CLI. Weitere Informationen finden Sie unter Installieren der Tanzu-CLI zur Verwendung mit TKG-Dienst-Clustern.

2 Stellen Sie eine Verbindung mit Supervisor her, indem Sie folgenden Befehl ausführen.

```
tanzu context create context_name --endpoint https://10.73.27.32
```

Dabei gilt:

- Der Wert von `context_name` ist der Name eines OIDC, dem Zugriff gewährt wird.
- Der Wert von `--endpoint` ist die IP-Adresse der Supervisor-Steuerungsebene.

Hinweis Hängen Sie bei der Fehlerbehebung `--stderr-only` an den Befehl an, zum Beispiel: `tanzu login --endpoint https://IP --name USER --stderr-only`.

3 Rufen Sie den Link in Ihrem Browser auf, sobald die Aufforderung ausgegeben wurde.

Abbildung 4-15. Tanzu-CLI-Anmeldung

4 Kopieren Sie den Autorisierungscode und fügen Sie ihn in die Tanzu-CLI ein.

```
tanzu context create context_name --endpoint https://10.73.27.32

Detected a vSphere Supervisor being used
Log in by visiting this link:
...
https://10.27.62.33/wcp/pinniped/oauth2/authorize?..
...
Optionally, paste your authorization code:
G2TcS145Q4e6A1YKf743n3BJlfQAQ_UdjXy38TtEEIo.ju4QV3PTsUvOigVUtQllZ7AJFU0YnjuLHTRVoNxvdZc
```

```
...
✔ successfully logged in to management cluster using the kubeconfig oidc-user
Checking for required plugins...
All required plugins are already installed and up-to-date
```

5 Nach der Authentifizierung können Sie die Tanzu-CLI verwenden, um einen TKG-Cluster im zielseitigen vSphere-Namespace bereitzustellen, auf den Sie zugreifen können. Weitere Informationen finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe der Tanzu-CLI.

Herstellen einer Verbindung zu einem TKG-Cluster als OIDC-Benutzer mit der Tanzu-CLI

Stellen Sie mithilfe der Tanzu-CLI eine Verbindung zum TKG-Cluster her und authentifizieren Sie sich mit Ihrem OIDC-Anbieter.

Voraussetzungen

Bei diesen Anweisungen wird davon ausgegangen, dass Supervisor mit einem unterstützten externen Identitätsanbieter (IDP) konfiguriert ist, dass Sie (als DevOps-Benutzer) mithilfe der Tanzu-CLI eine Verbindung zu Supervisor hergestellt und einen TKG-Cluster bereitgestellt haben. Beschäftigen Sie sich gegebenenfalls mit folgendem Thema:

- Herstellen einer Verbindung zu TKG-Clustern auf Supervisor mithilfe eines externen Identitätsanbieters
- Kapitel 7 Bereitstellen von TKG-Dienstclustern

Workflow für DevOps-Benutzer

Als DevOps-Benutzer mit Bearbeitungsberechtigungen im Ziel-vSphere-Namespace verwenden Sie die Tanzu-CLI, um eine gemeinsam nutzbare Datei vom Typ `kubeconfig` zu erzeugen, die Sie dann an TKG-Clusterbenutzer verteilen. In Kubernetes enthält ein Konfigurationskontext einen Cluster, einen Namespace und einen Benutzer. Sie können sich den Clusterkontext in der Datei `.kube/config` ansehen. Diese Datei wird gemeinhin als `kubeconfig`-Datei bezeichnet.

Hinweis Diese Schritte müssen von einem DevOps-Benutzer mit Bearbeitungsberechtigungen im Ziel-vSphere-Namespace durchgeführt werden.

1 Stellen Sie sicher, dass der Tanzu-CLI-Kontext auf Supervisor festgelegt ist.

   Weitere Informationen hierzu finden Sie unter Herstellen einer Verbindung zu Supervisor mit der Tanzu-CLI und einem externen IDP.

2 Stellen Sie sicher, dass der vSphere-Administrator Benutzerberechtigungen für den Ziel-vSphere-Namespace konfiguriert hat.

   Externe OIDC-Benutzer und -Gruppen werden vSphere-Namespace-Rollen direkt zugeordnet. Clusterbenutzer sollten dem vSphere-Namespace hinzugefügt werden, bevor Sie die gemeinsam nutzbare Kubeconfig-Datei erzeugen.

Weitere Informationen hierzu finden Sie unter Konfigurieren von vSphere-Namespace-Berechtigungen für Benutzer und Gruppen externer Identitätsanbieter.

3 Listet die im Ziel-vSphere-Namespace bereitgestellten TKG-Cluster auf.

```
tanzu cluster list --namespace VSPHERE-NAMESPACE
```

4 Erzeugen Sie eine gemeinsam nutzbare Kubeconfig-Datei für den TKG-Zielcluster.

```
tanzu cluster kubeconfig get CLUSTER-NAME --namespace=NAMESPACE
```

5 Verteilen Sie die gemeinsam genutzte Kubeconfig-Datei an Clusterbenutzer, damit diese sich beim TKG-Cluster anmelden können.

Workflow für Clusterbenutzer

Führen Sie diese Schritte aus, um sich als Clusterbenutzer bei einem TKG-Cluster anzumelden.

1 Rufen Sie die Kubeconfig-Datei beim DevOps-Benutzer ab.

2 Melden Sie sich mit der kubeconfig-Datei und kubectl beim TKGS-Cluster an.

```
kubectl --kubeconfig
```

3 Schließen Sie die Authentifizierung beim Browser ab.

   a Rufen Sie den Link in Ihrem Browser auf, sobald die Aufforderung ausgegeben wurde.

   b Kopieren Sie den Autorisierungscode und fügen Sie ihn in die CLI ein.

4 Verwenden Sie `kubectl` für die Interaktion mit dem Cluster.

Herstellen einer Verbindung zu TKG-Dienst-Clustern als Kubernetes-Administrator und Systembenutzer

Sie können eine Verbindung zu TKG-Dienst-Clustern als Kubernetes-Administrator und als Systembenutzer herstellen, um die Verwaltung und Fehlerbehebung von TKG-Dienst-Clustern zu unterstützen.

Herstellen einer Verbindung mit der TKG-Dienst-Steuerungsebene als Kubernetes-Administrator

Sie können als `kubernetes-admin`-Benutzer eine Verbindung mit der Steuerungsebene des TKG-Dienst-Clusters herstellen, um administrative Aufgaben durchzuführen und Clusterprobleme zu beheben.

Eine gültige `kubeconfig`-Datei für einen bereitgestellten Tanzu Kubernetes-Cluster ist im Supervisor als geheimes Objekt mit dem Namen *`TKG-CLUSTER-NAME`*`-kubeconfig` verfügbar. Mit diesem geheimen Schlüssel können Sie als Benutzer `kubernetes-admin` eine Verbindung mit der Cluster-Steuerungsebene herstellen.

Verfahren

1 Stellen Sie eine Verbindung zu Supervisor her.

2 Ändern Sie den Kontext in den vSphere-Namespace, in dem der TKG-Zielcluster bereitgestellt wird.

```
kubectl config use-context VSPHERE-NAMESPACE
```

3 Zeigen Sie die geheimen Objekte im Namespace an.

```
kubectl get secrets
```

Das geheime Objekt hat den Namen *TKG-CLUSTER-NAME*-kubeconfig.

```
kubectl config use-context tkg-cluster-ns
Switched to context "tkg-cluster-ns".
ubuntu@ubuntu:~$ kubectl get secrets
NAME                                 TYPE                                  DATA   AGE
...
tkg-cluster-1-kubeconfig            Opaque                                1      23h
...
```

4 Entschlüsseln Sie das geheime Objekt, indem Sie den folgenden Befehl ausführen.

Der geheime Schlüssel ist Base64-verschlüsselt. So entschlüsseln Sie den Schlüssel: Verwenden Sie unter Linux `base64 --decode` (oder `base64 -d`). Verwenden Sie unter macOS `base64 --Decode` (oder `base64 -D`). Verwenden Sie unter Windows ein Online-Tool.

```
kubectl get secret TKG-CLUSTER-NAME-kubeconfig -o jsonpath='{.data.value}' | base64 -d >
tkgs-cluster-kubeconfig-admin
```

Dieser Befehl entschlüsselt das geheime Objekt und schreibt es in eine lokale Datei namens `tkgs-cluster-kubeconfig-admin`. Überprüfen Sie mithilfe des Befehls `cat` den Dateiinhalt.

5 Stellen Sie als Kubernetes-Administrator mithilfe der entschlüsselten `tkg-cluster-kubeconfig-admin`-Datei eine Verbindung zum TKG-Cluster her.

Dazu haben Sie zwei Optionen:

Option	Beschreibung
--kubeconfig <path\to\kubeconfig>	Verwenden Sie das Flag `--kubeconfig` und den Pfad zur lokalen kubeconfig-Datei. Angenommen etwa, die kubeconfig-Datei befindet sich im selben Verzeichnis, in dem Sie den folgenden Befehl ausführen: `kubectl --kubeconfig tkg-cluster-kubeconfig-admin get nodes`.
KUBECONFIG	Legen Sie die Umgebungsvariable KUBECONFIG so fest, dass Sie auf die entschlüsselte kubeconfig-Datei verweist, und führen Sie kubectl aus, z. B. als `kubectl get nodes`.

Sie sollten die Knoten im Cluster sehen.

6 Wenn Sie sich als DevOps-Benutzer mit Bearbeitungsberechtigungen für den vSphere-Namespace mit der Tanzu-CLI bei einem TKG-Cluster als Admin-Benutzer anmelden möchten, führen Sie folgenden Befehl aus:

```
tanzu cluster kubeconfig get CLUSTER-NAME --admin
```

Dieser Befehl erzeugt eine Datei vom Typ `kubeconfig` mit dem Zertifikat/privaten Schlüssel für die Kubernetes-Steuerungsebene (unter Umgehung der gesamten Autorisierung). Sie können sich dann mit dieser `kubeconfig`-Datei beim Cluster anmelden. Weitere Informationen hierzu finden Sie unter #unique_36.

Herstellen einer Verbindung zu TKG-Dienst-Clusterknoten mit SSH als Systembenutzer unter Verwendung eines Privatschlüssels

Als `vmware-system-user` können Sie eine Verbindung zu einem TKG-Clusterknoten über SSH unter Verwendung eines Privatschlüssels herstellen.

Sie können sich über SSH bei jedem TKG-Clusterknoten als `vmware-system-user`-Benutzer anmelden. Der geheime Schlüssel, der den privaten SSH-Schlüssel enthält, wird `CLUSTER-NAME-ssh` genannt. Weitere Informationen finden Sie unter Abrufen von geheimen Schlüsseln für TKG-Cluster mithilfe von Kubectl.

Um eine Verbindung mit einem TKG-Clusterknoten über SSH mithilfe eines privaten Schlüssels herzustellen, erstellen Sie einen Jump-Box-vSphere Pod im Supervisor.

Voraussetzungen

In dieser Aufgabe stellen Sie einen vSphere Pod als Jump-Host für SSH-Konnektivität bereit. vSphere-Pods erfordert NSX-Netzwerk für Supervisor. Wenn Sie vDS-Netzwerke für Supervisor verwenden, finden Sie weitere Informationen unter Herstellen einer Verbindung zu TKG-Dienst-Clusterknoten mit SSH als Systembenutzer unter Verwendung eines Kennworts.

Verfahren

1 Stellen Sie eine Verbindung zu Supervisor her.

Weitere Informationen hierzu finden Sie unter Herstellen einer Verbindung zu Supervisor als vCenter Single Sign-On-Benutzer mit Kubectl.

2 Erstellen Sie eine Umgebungsvariable namens **`NAMESPACE`**, deren Wert der Name des vSphere-Namespace ist, in dem der TKG-Zielcluster bereitgestellt wird.

```
export NAMESPACE=VSPHERE-NAMESPACE
```

3 Wechseln Sie den Kontext zum vSphere-Namespace, in dem der Tanzu Kubernetes-Cluster bereitgestellt wird.

```
kubectl config use-context $NAMESPACE
```

4 Zeigen Sie das geheime Objekt *TKG-CLUSTER-NAME*-ssh an.

```
kubectl get secrets
```

5 Erstellen eines geheimen Anmeldedatenschlüssels für die Docker Hub-Registrierung

Standardmäßig wird das zum Erstellen des vSphere-Pods (PhotonOS) verwendete Image aus Docker Hub abgerufen. Unter Umständen benötigen Sie einen geheimen Anmeldedatenschlüssel, um das Image erfolgreich abzurufen. Weitere Informationen hierzu finden Sie unter Erstellen eines geheimen Schlüssels für die private Registrierung.

6 Erstellen Sie einen vSphere Pod mithilfe der folgenden `jumpbox.yaml`.

Ersetzen Sie den `namespace`-Wert `YOUR-NAMESPACE` durch den vSphere-Namespace, in dem der Zielcluster bereitgestellt wird. Ersetzen Sie den `secretName`-Wert `YOUR-CLUSTER-NAME`-ssh durch den Namen des Zielclusters.

```
apiVersion: v1
kind: Pod
metadata:
  name: jumpbox
  namespace: YOUR-NAMESPACE     #REPLACE
spec:
  containers:
  - image: "photon:3.0"
    name: jumpbox
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "yum install -y openssh-server; mkdir /root/.ssh; cp /root/ssh/ssh-privatekey /
root/.ssh/id_rsa; chmod 600 /root/.ssh/id_rsa; while true; do sleep 30; done;" ]
    volumeMounts:
      - mountPath: "/root/ssh"
        name: ssh-key
        readOnly: true
    resources:
      requests:
        memory: 2Gi
  volumes:
    - name: ssh-key
      secret:
        secretName: YOUR-CLUSTER-NAME-ssh     #REPLACE
  imagePullSecrets:
    - name: regcred
```

7 Stellen Sie den Pod durch Anwenden der `jumpbox.yaml`-Spezifikation bereit.

```
kubectl apply -f jumpbox.yaml
```

```
pod/jumpbox created
```

8 Überprüfen Sie, ob der Pod ausgeführt wird.

```
kubectl get pods
```

```
NAME      READY   STATUS    RESTARTS   AGE
jumpbox   1/1     Running   0          3h9m
```

Hinweis Sie sollten auch den Jumpbox-Pod in vCenter im vSphere-Namespace sehen.

9 Erstellen Sie eine Umgebungsvariable mit der IP-Adresse des Zielclusterknotens, indem Sie den folgenden Befehlssatz ausführen.

a Rufen Sie den Namen der virtuellen Zielmaschine ab.

```
kubectl get virtualmachines
```

b Erstellen Sie die Umgebungsvariable `VMNAME`, deren Wert der Name des Zielknotens ist.

```
export VMNAME=NAME-OF-THE-VIRTUAL-MACHINE
```

c Erstellen Sie die Umgebungsvariable `VMIP`, deren Wert die IP-Adresse der Zielknoten-VM ist.

```
export VMIP=$(kubectl -n $NAMESPACE get virtualmachine/$VMNAME -o
jsonpath='{.status.vmIp}')
```

10 Greifen Sie mithilfe des Jump Box-Pods per SSH auf den Clusterknoten zu, indem Sie den folgenden Befehl ausführen.

```
kubectl exec -it jumpbox  /usr/bin/ssh vmware-system-user@$VMIP
```

Wichtig Es dauert ca. 60 Sekunden, um den Container zu erstellen und die Software zu installieren. Wenn Sie die Meldung „error executing command in container: container_linux.go:370: starting container process caused: exec: "/usr/bin/ssh": stat /usr/bin/ssh: no such file or directory“ erhalten, führen Sie den Befehl in ein paar Sekunden erneut aus.

11 Bestätigen Sie die Echtheit des Hosts, indem Sie **Ja** eingeben.

```
The authenticity of host '10.249.0.999 (10.249.0.999)' can't be established.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.249.0.999' (ECDSA) to the list of known hosts.
Welcome to Photon 3.0
```

12 Bestätigen Sie, dass Sie beim Zielknoten als `vmware-system-user` angemeldet sind.

Beispielsweise gibt die folgende Ausgabe an, dass Sie als Systembenutzer bei einem Knoten für die Steuerungsebene angemeldet sind.

```
vmware-system-user@tkg-cluster-1-control-plane-66tbr [ ~ ]$
```

13 Führen Sie die gewünschten Vorgänge auf dem Knoten aus.

Achtung Nach der Authentifizierung müssen Sie möglicherweise `sudo` oder `sudo su` verwenden, um bestimmte Vorgänge auf dem Knoten durchzuführen, wie z. B. den Neustart von kubelet.

14 Wenn Sie fertig sind, geben Sie **`exit`** ein, um sich von der SSH-Sitzung auf dem vSphere Pod abzumelden.

15 Um den Pod zu löschen, führen Sie den Befehl `kubectl delete pod jumpbox` aus.

Vorsicht Aus Sicherheitsgründen sollten Sie nach Abschluss des Vorgangs den Jumpbox-Pod löschen. Bei Bedarf können Sie ihn zu einem späteren Zeitpunkt neu erstellen.

Herstellen einer Verbindung zu TKG-Dienst-Clusterknoten mit SSH als Systembenutzer unter Verwendung eines Kennworts

Als `vmware-system-user` können Sie eine Verbindung zu einem Arbeitslast-Clusterknoten mit SSH herstellen.

Sie können als `vmware-system-user`-Benutzer mit einem Kennwort eine Verbindung mit einem Clusterknoten herstellen. Das Kennwort wird als geheimer Schlüssel mit dem Namen `CLUSTER-NAME-ssh-password` gespeichert. Das Kennwort ist in `.data.ssh-passwordkey` Base64-verschlüsselt. Sie können das Kennwort über eine SSH-Sitzung angeben. Weitere Informationen finden Sie unter Abrufen von geheimen Schlüsseln für TKG-Cluster mithilfe von Kubectl.

Voraussetzungen

Zum Weiterleiten von SSH-Verbindungen an das entsprechende Arbeitslastnetzwerk stellen Sie eine Linux-Jump-Host-VM in der vSphere-Umgebung bereit, in der **Arbeitslastverwaltung** aktiviert ist. Weitere Informationen hierzu finden Sie unter Erstellen einer VM für einen Linux-Jump-Host.

Hinweis Die Bereitstellung einer Jump-Host-VM ist eine strenge Anforderung, wenn Sie ein vDS-Netzwerk verwenden und eine Verbindung mit Clusterknoten über SSH herstellen möchten. Sie können diesen Ansatz auch mit einem NSX-Netzwerk verwenden, wenn Sie statt eines privaten Schlüssels lieber ein Kennwort zum Herstellen einer Verbindung über SSH verwenden möchten.

Verfahren

1. Rufen Sie die IP-Adresse der Jump-Host-VM, den Benutzernamen und das Kennwort ab.

   Weitere Informationen hierzu finden Sie unter Erstellen einer VM für einen Linux-Jump-Host.

2. Stellen Sie eine Verbindung mit dem Supervisor her.

   Weitere Informationen hierzu finden Sie unter Herstellen einer Verbindung zu Supervisor als vCenter Single Sign-On-Benutzer mit Kubectl.

3. Ändern Sie den Kontext in den vSphere-Namespace, in dem der TKG-Zielcluster bereitgestellt wird.

   ```
   kubectl config use-context VSPHERE-NAMESPACE
   ```

4. Rufen Sie die IP-Adresse des Zielclusterknotens ab.

   Erstellen Sie eine Liste der Knoten.

   ```
   kubectl get virtualmachines
   ```

   Beschreiben Sie die Knoten, um die IP-Adresse des Zielknotens abzurufen.

   ```
   kubectl describe virtualmachines
   ```

5. Zeigen Sie den geheimen `TKG-CLUSTER-NAME-ssh-password`-Schlüssel an.

   ```
   kubectl get secrets
   ```

6. Rufen Sie den SSH-Kennwortschlüssel für den Zielcluster ab.

   ```
   kubectl get secrets TKG-CLUSTER-NAME-ssh-password -o yaml
   ```

   Zum Beispiel wird der SSH-Kennwortschlüssel zurückgegeben.

   ```
   apiVersion: v1
   data:
     ssh-passwordkey: RU1pQllLTC9TRjVFV0RBcCtmd1zwOTROeURYSWNGeXNReXJhaXRBUllYaz0=
   ```

7. Entschlüsseln Sie den SSH-Kennwortschlüssel.

   Der geheime Schlüssel ist Base64-verschlüsselt. So entschlüsseln Sie den Schlüssel: Verwenden Sie unter Linux `base64 --decode` (oder `base64 -d`). Verwenden Sie unter macOS `base64 --Decode` (oder `base64 -D`). Verwenden Sie unter Windows ein Online-Tool.

   ```
   echo <ssh-passwordkey> | base64 --decode
   ```

8. Stellen Sie als `vmware-system-user` eine Verbindung zum Zielclusterknoten über SSH her.

   ```
   ssh vmware-system-user@TKG-CLUSTER-NODE-IP-ADDRESS
   ```

9 Melden Sie sich mit dem entschlüsselten Kennwort an.

Erstellen einer VM für einen Linux-Jump-Host

Um SSH mithilfe eines Kennworts für Arbeitslast-Clusterknoten zu nutzen, erstellen Sie eine Jump-Box-VM, die eine Verbindung zum Arbeitslastnetzwerk und der Verwaltung oder zum Front-End-Netzwerk für SSH-Tunnel herstellt.

Erstellen einer VM für einen Linux-Jump-Host

Führen Sie diese Schritte aus, um eine Linux-Jump-Box-VM zu erstellen. Dazu haben Sie mehrere Möglichkeiten. Dies ist ein Ansatz. Die Anweisungen verwenden PhotonOS und können hier heruntergeladen werden: https://github.com/vmware/photon/wiki/Downloading-Photon-OS.

Hinweis Diese Methode zum Erstellen eines Jump-Hosts ist für vDS-Netzwerkumgebungen vorgesehen. Wenn Sie NSX verwenden, erstellen Sie einen Jump-Host mit einem vSphere Pod. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu TKG-Dienst-Clusterknoten mit SSH als Systembenutzer unter Verwendung eines Privatschlüssels.

1 Melden Sie sich bei vCenter Server mithilfe des vSphere Clients an.

2 Erstellen Sie eine neue virtuelle Maschine.

3 Wählen Sie das Linux-Gastbetriebssystem aus, in diesem Beispiel VMware Photon OS (64 Bit).

4 Installieren Sie das Betriebssystem. Laden Sie hierzu die ISO-Datei herunter, hängen Sie sie an die VM an und starten Sie sie.

5 Konfigurieren Sie die VM mit einer IP-Adresse im Arbeitslastnetzwerk > Namespace-Netzwerk.

   Hinweis Es ist möglich, einen IP-Konflikt zu erstellen, wenn der Geltungsbereich des Arbeitslastnetzwerkbereichs den gesamten Netzwerkbereich der verwendeten Portgruppe belegt.

6 Fügen Sie der VM eine zweite virtuelle Netzwerkkarte hinzu und weisen Sie sie dem Management- oder Frontend-Netzwerk zu.

7 Schließen Sie die Konfiguration des Betriebssystems ab und schalten Sie die VM nach dem Neustart ein.

8 Melden Sie sich bei der vSphere-Konsole für die VM als Root-Benutzer an.

9 Erstellen Sie eine Netzwerkschnittstelle für die neue Netzwerkkarte und weisen Sie ihr eine IP-Adresse im Frontend-Netzwerk zu.

```
ifconfig eth1 IP-ADDRESS netmask NETMASK up
```

   Hinweis Diese Methode ist bei Neustarts nicht dauerhaft.

10 Stellen Sie sicher, dass Sie das Gateway und den DNS-Server über diese Schnittstelle anpingen können.

11 Richten Sie in der vSphere-Konsole für die VM einen SSH-Benutzer mit Zertifikaten ein. Überprüfen Sie, ob es funktioniert, indem Sie eine verschachtelte Shell erstellen.

12 Melden Sie sich per SSH über das Frontend-Netzwerk als SSH-Benutzer bei der Jump-Box an, um zu überprüfen, ob dies funktioniert.

13 Installieren Sie sshpass auf der VM (damit Sie sich über SSH mithilfe eines Kennworts anmelden können). Für PhotonOS lautet der Befehl folgendermaßen:

```
tdnf install -y sshpass
```

14 Fügen Sie den öffentlichen Schlüssel des Clients zur Datei `~/.ssh/authorized_keys` hinzu und starten Sie den `sshd`-Prozess neu, damit SSH ohne Kennwort ausgeführt werden kann.

- Rufen Sie Ihren öffentlichen Schlüssel ab, z. B.: `cat ~/.ssh/id_rsa.pub`.
- Greifen Sie auf die Jump-Host-VM zu.
- Erstellen Sie das SSH-Verzeichnis (sofern nicht vorhanden): `mkdir -p ~/.ssh`.
- Hängen Sie den öffentlichen Schlüssel an die Datei `authorized_keys` an: `echo ssh-rsa AAAA.... >> ~/.ssh/authorized_keys`. Ersetzen Sie `ssh-rsa AAAA....` durch die gesamte öffentliche Schlüsselzeichenfolge, die durch den Befehl `cat ~/.ssh/id_rsa.pub` ausgegeben wurde.
- Stellen Sie sicher, dass das Verzeichnis `~/.ssh` und die Datei `authorized_keys` über die entsprechenden Berechtigungen verfügen, z. B. `chmod -R go= ~/.ssh`.

Erstellen einer dedizierten Gruppe und Rolle für Plattformoperatoren

Eine optionale Möglichkeit, vSphere IaaS control plane-Operatoren Berechtigungen zuzuweisen, einschließlich Personen, die für den Betrieb von Supervisor- und TKG-Dienst-Clustern verantwortlich sind, besteht darin, eine dedizierte vSphere Benutzergruppe und -rolle speziell für solche Operatoren zu erstellen.

Informationen zur Plattformoperatoren-Gruppe und -Rolle

Als Best Practice für die Sicherheit sollten Sie die Zuweisung der vSphere-Administratorrolle an Plattformoperatoren vermeiden, da diese Rolle mehr Berechtigungen gewährt, als für den Betrieb von TKG-Dienst-Clustern erforderlich sind. Nach dem Prinzip der „geringstmöglichen Berechtigungen“ können Sie eine dedizierte Benutzergruppe, ein Dienstkonto (Benutzer) und eine benutzerdefinierte Rolle für die Verwendung mit Plattformoperatoren erstellen und der Benutzergruppe dann benutzerdefinierte Rollenberechtigungen für vSphere-Objekte erteilen.

Hinweis Sie müssen bei vCenter Server als vSphere-Administrator angemeldet sein, um alle Aufgaben in diesem Thema durchführen zu können.

Hinweis vSphere Gruppen- und Rollennamen sind benutzerdefinierte Zeichenfolgen. Bei den hier angegebenen Namen handelt es sich um Beispiele, die Sie entsprechend Ihren Sicherheits- und Geschäftsanforderungen übernehmen, anpassen oder ändern können.

Warnung Sie müssen die hier bereitgestellten Beispielrollenberechtigungen im Zusammenhang mit Ihren Sicherheits- und Geschäftsanforderungen bewerten, die Rolle testen, um Konformität zu gewährleisten, und sie entsprechend anpassen. Möglicherweise sind nicht alle hier verwendeten Berechtigungen für Ihre Anforderungen geeignet, und Sie benötigen möglicherweise zusätzliche Berechtigungen. Eine vollständige Liste der vSphere-Berechtigungen und -Sicherheitsüberlegungen finden Sie in der Dokumentation zur vSphere-Sicherheit.

Teil 1: Erstellen einer Plattformoperatoren-Gruppe und eines -Benutzers

Erstellen Sie in vCenter oder einem AD/LDAP-System, das in vCenter integriert ist, die Plattformoperatoren-Gruppe und ein anfängliches Benutzerkonto.

1 Melden Sie sich über den vSphere-Client als Administrator bei vCenter Server an.

2 Gehen Sie zu **Verwaltung > Single Sign On > Benutzer und Gruppen**.

3 Wählen Sie die Registerkarte **Gruppen** aus.

4 Klicken Sie auf **Hinzufügen** und erstellen Sie eine neue Gruppe:

- Name: *platform-operators-group*
- Beschreibung: *Gruppenkonto für Kubernetes-Operatoren von Supervisor- und TKG-Dienstclustern*
- Klicken Sie auf **Hinzufügen**.

5 Wählen Sie die Registerkarte **Benutzer** aus.

6 Klicken Sie auf **Hinzufügen** und erstellen Sie zu Testzwecken einen neuen Benutzer.

- Name: *platform-operator-00*
- Kennwort: Geben Sie ein sicheres Kennwort ein, das den Anforderungen entspricht.

- Klicken Sie auf **Hinzufügen**.

7 Wählen Sie die Registerkarte **Gruppen** aus.

8 Fügen Sie der Gruppe den neuen Benutzer hinzu.

- Wählen Sie die Gruppe *platform-operators-group* aus.
- Klicken Sie auf **Mitglieder hinzufügen**.
- Wählen Sie **vsphere.local** aus.
- Suchen Sie nach dem Benutzernamen *platform-operator-00*.
- Wählen Sie diesen Benutzer aus und klicken Sie auf **Hinzufügen**.
- Klicken Sie auf **Speichern**.

Teil 2: Hinzufügen der Plattformoperatoren-Gruppe zur Dienstanbieter-Benutzergruppe

Fügen Sie die neue Plattformoperatoren-Gruppe zur Dienstanbieter-Benutzergruppe hinzu. Auf diese Weise haben Mitglieder der Plattformoperatoren-Gruppe die Möglichkeit, vSphere-Namespaces auf dem Bildschirm vCenter **Bestand > Hosts und Cluster** anzuzeigen.

1 Gehen Sie zu **Verwaltung > Single Sign On > Benutzer und Gruppen**.

2 Wählen Sie die Registerkarte **Gruppen** aus.

3 Suchen Sie die Gruppe **ServiceProviderUsers**.

4 Bearbeiten Sie die Gruppe **ServiceProviderUsers** und fügen Sie die **platform-operators-group** als Mitglied hinzu.

5 Klicken Sie auf **Speichern**.

Teil 3: Erstellen der Plattformoperatoren-Rolle

Erstellen Sie eine benutzerdefinierte vCenter SSO-Rolle für Plattformoperatoren.

Hinweis Die Rolle umfasst alle erforderlichen Berechtigungen zum Bereitstellen und Betreiben von Supervisor und TKG-Dienst-Clustern, einschließlich der Verwaltung von Inhaltsbibliotheken. Möglicherweise müssen Sie die dieser Rolle zugewiesenen Berechtigungen basierend auf Ihrem Geschäft und Ihren Sicherheitsanforderungen anpassen. Darüber hinaus müssen Sie die Rolle testen, um sicherzustellen, dass sie Ihre Anforderungen erfüllt.

1 Navigieren Sie mithilfe von vSphere Client zu **Verwaltung > Zugriffssteuerung > Rollen**.

2 Wählen Sie **Neu** aus und erstellen Sie eine neue Rolle namens **`platform-operators-role`**.

3 Definieren Sie die folgenden Berechtigungen für diese Rolle.

4 Klicken Sie anschließend auf **Speichern**.

```
- Alarms
  - Acknowledge alarm &
```

```
  - Create alarm &
  - Disable alarm action on entity &
  - Modify alarm &
  - Remove alarm &
  - Set alarm status &
- Certificate Authority
  - Create/Delete (below Admins priv) &
- Certificate Management
  - Create/Delete (below Admins priv) &
- Cns
  - Searchable * &
- Compute Policy
  - Create and Delete Compute Policy &
- Content Library
  - Add library item &
  - Check in a template &
  - Check out a template &
  - Create local library &
  - Create subscribed library &
  - Delete library item &
  - Delete local library &
  - Delete subscribed library &
  - Download files &
  - Evict library item &
  - Evict subscribed library &
  - Import storage &
  - Probe subscription information &
  - Read storage &
  - Sync library item &
  - Sync subscribed library &
  - Type introspection &
  - Update configuration settings &
  - Update library &
  - Update library item &
  - Update local library &
  - Update subscribed library &
  - View configuration settings &
- Datastore
  - Allocate space * & $
  - Browse datastore * &
  - Configure datastore &
  - Low level file operations * &
  - Remove file &
  - Rename datastore &
  - Update virtual machine files &
  - Update virtual machine metadata &
- Extension
  - Register extension &
  - Unregister extension &
  - Update extension &
- Folder
  - Create folder &
  - Delete folder &
  - Move folder &
  - Rename folder &
```

```
- Global
  - Cancel task &
  - Disable methods * &
  - Enable methods * &
  - Global tag &
  - Health &
  - Licenses * &
  - Log event &
  - Manage custom attributes &
  - Service managers &
  - Set custom attribute &
  - System tag &
- Host
  - Configuration
    - Network configuration $
- Host profile
  - View &
- Hybrid Linked Mode
  - Manage &
- Namespaces
  - Modify cluster-wide configuration
  - Modify cluster-wide namespace self-service configuration
  - Modify namespace configuration
- Network
  - Assign network * & $
- Resource
  - Apply recommendation &
  - Assign vApp to resource pool * &
  - Assign virtual machine to resource pool &
  - Create resource pool &
  - Modify resource pool &
  - Move resource pool &
  - Query vMotion &
  - Remove resource pool &
  - Rename resource pool &
- Scheduled task
  - Create tasks &
  - Modify task &
  - Remove task &
  - Run task &
- Sessions
  - Message * &
  - Validate session * &
- VM storage policies
  - View VM storage policies *
- Storage views
  - View &
- Supervisor Services
  - Manage Supervisor Services
- Trusted Infrastructure administrator
  - Manage Trusted Infrastructure Hosts &
- vApp
  - Add virtual machine &
  - Assign resource pool &
  - Assign vApp &
```

```
    - Clone &
    - Create &
    - Delete &
    - Export &
    - Import * $
    - Move &
    - Power off &
    - Power on &
    - Rename &
    - Suspend &
    - Unregister &
    - View OVF environment &
    - vApp application configuration &
    - vApp instance configuration &
    - vApp managedBy configuration &
    - vApp resource configuration &
  - Virtual machine
    - Change Configuration
      - Acquire disk lease &
      - Add existing disk * & $
      - Add new disk * &
      - Add or remove device * &
      - Advanced configuration * & $
      - Change CPU count * &
      - Change Memory * &
      - Change Settings * &
      - Change Swapfile placement &
      - Change Resource &
      - Configure Host USB device &
      - Configure Raw device * &
      - Configure managedBy &
      - Display connection settings &
      - Extend virtual disk * &
      - Modify device settings * &
      - Query Fault Tolerance compatibility &
      - Query unowned files &
      - Reload from path &
      - Remove disk * &
      - Rename &
      - Reset guest information &
      - Set annotation &
      - Toggle disk change tracking * &
      - Upgrade virtual machine compatibility &
    - Edit Inventory
      - Create from existing * &
      - Create new &
      - Move &
      - Remove * &
      - Register &
      - Unregister &
    - Guest operations
      - Guest operation alias modification &
      - Guest operation alias query &
      - Guest operation modifications &
      - Guest operation program execution &
```

```
    - Guest operation queries &
  - Interaction
    - Answer question &
    - Backup operation on virtual machine &
    - Configure CD media &
    - Configure floppy media &
    - Connect devices &
    - Console interaction &
    - Create screenshot &
    - Defragment all disks &
    - Drag and drop &
    - Guest operating system management by VIX API &
    - Inject USB HID scan codes &
    - Install VMware Tools &
    - Pause or Unpause &
    - Power off * &
    - Power on * &
    - Reset &
    - Suspend &
  - Provisioning
    - Allow disk access &
    - Allow file access &
    - Allow read-only disk access * &
    - Allow virtual machine download * &
    - Allow virtual machine files upload &
    - Clone template &
    - Clone virtual machine &
    - Create template from virtual machine &
    - Customize guest &
    - Deploy template * &
    - Mark as template &
    - Mark as virtual machine &
    - Modify customization specification &
    - Promote disks &
    - Read customization specifications &
  - Service configuration
    - Allow notifications &
    - Allow polling of global event notifications &
    - Manage service configurations &
    - Modify service configuration &
    - Query service configurations &
    - Read service configuration &
  - Snapshot management
    - Create snapshot * &
    - Remove snapshot * &
    - Rename snapshot &
    - Revert to snapshot &
  - vSphere Replication
    - Configure replication &
    - Manage replication &
    - Monitor replication &
- Virtual Machine Classes
  - Manage Virtual Machine Classes
- vSan
  - Cluster &
```

```
    - ShallowRekey &
- vService
  - Create dependency &
  - Destroy dependency &
  - Reconfigure dependency configuration &
  - Update dependency &
- vSphere Tagging
  - Assign or Unassign vSphere Tag &
  - Assign or Unassign vSphere Tag on Object &
  - Create vSphere Tag &
  - Create vSphere Tag Category &
  - Delete vSphere Tag &
  - Delete vSphere Tag Category &
  - Edit vSphere Tag &
  - Edit vSphere Tag Category &
  - Modify UsedBy Field For Category &
  - Modify UsedBy Field For Tag &
```

Teil 4: Zuweisen von vCenter-Objektberechtigungen zur Plattformoperatoren-Gruppe und -Rolle

Weisen Sie der Plattformoperatoren-Gruppe Berechtigungen für die vCenter-Objekte zu, die Supervisor und TKG-Dienst-Cluster verwenden.

1 Wählen Sie in vCenter die Ansicht **Bestandsliste** aus.

2 Klicken Sie für jedes der unten aufgeführten vCenter-Objekte mit der rechten Maustaste auf das Objekt und wählen Sie **Berechtigung hinzufügen** aus.

3 Wählen Sie als Benutzer/Gruppe die Gruppe **platform-operators-group** aus.

4 Wählen Sie als Rolle die Rolle **platform-operators-group-role** aus.

5 Für einige Objekte müssen Sie wie angegeben **An untergeordnete Objekte weitergeben** auswählen.

- **Hosts und Cluster**
  - Das Root-vCenter-Serverobjekt.
  - Das Datencenter sowie alle Host- und Clusterordner, vom Datencenter-Objekt bis zum Cluster, der die TKG-Bereitstellung verwaltet.
  - Der vCenter-Zielcluster, in dem der Supervisor und **An untergeordnete Objekte weitergeben** aktiviert ist (für die ESXi-Hosts usw.).
  - Zielressourcenpools, für die **An untergeordnete Objekte weitergeben** aktiviert ist.
- **VMs und Vorlagen**
  - Das Datencenterobjekt auf oberster Ebene, bei dem **An untergeordnete Objekte weitergeben** aktiviert ist.
  - Alternativ für mehr Granularität die Ziel-VM und die Vorlagenordner mit aktivierter Option **An untergeordnete Objekte weitergeben**.

- **Speicher**
  - Das Datencenterobjekt auf oberster Ebene, bei dem **An untergeordnete Objekte weitergeben** aktiviert ist.
  - Alternativ das freigegebene Datenspeicherobjekt (z. B. vsanDatastore) ohne aktivierte Option **An untergeordnete Objekte weitergeben** oder einzelne Datenspeicher und alle Speicherordner vom Datencenterobjekt bis zu den Datenspeichern, die für TKG-Bereitstellungen verwendet werden, mit aktivierter Option **An untergeordnete Objekte weitergeben**.
- **Netzwerke**
  - Das Datencenterobjekt auf oberster Ebene, bei dem **An untergeordnete Objekte weitergeben** aktiviert ist.
  - Alternativ können einzelne Netzwerke, Distributed Switches und verteilte Portgruppen verwendet werden, denen Cluster zugewiesen werden.

Teil 5: Zuordnen der Plattformoperatoren-Gruppe und -Rolle

Weisen Sie der Plattformoperatoren-Gruppe und -Rolle Berechtigungen zu.

1 Navigieren Sie mit dem vSphere Client zu **Administration > Zugriffssteuerung > Globale Berechtigung > Berechtigung hinzufügen**.

2 Fügen Sie die Plattformoperatoren-Rolle zur Plattformoperatoren-Gruppe hinzu.
   - Klicken Sie auf **Hinzufügen**.
   - Wählen Sie als Domäne **vsphere.local** aus.
   - Geben Sie als Benutzer/Gruppe die Gruppe **platform-operators-group** ein.
   - Wählen Sie als Rolle die Rolle **platform-operators-role** aus.
   - Aktivieren Sie das Kontrollkästchen **An untergeordnete Objekte weitergeben**.
   - Klicken Sie auf **OK**, um die Gruppe mit den Berechtigungen der Inhaltsbibliothek zu aktualisieren.

3 Fügen Sie die Rolle vSphere Kubernetes-Manager zur Plattformoperatoren-Gruppe hinzu.
   - Klicken Sie auf **Hinzufügen**.
   - Wählen Sie als Domäne **vsphere.local** aus.
   - Geben Sie als Benutzer/Gruppe die Gruppe **platform-operators-group** ein.
   - Wählen Sie als Rolle die Rolle **vSphere Kubernetes Manager** aus.
   - Aktivieren Sie das Kontrollkästchen **An untergeordnete Objekte weitergeben**.
   - Klicken Sie auf **OK**, um die Gruppe mit den Berechtigungen der Inhaltsbibliothek zu aktualisieren.

Teil 6: Validieren der Plattformoperatoren-Gruppe und -Rolle

Testen Sie die neue Plattformoperatoren-Gruppe und -Rolle. Sie müssen sicherstellen, dass die Gruppe und die Rolle Ihre Sicherheits- und Geschäftsanforderungen erfüllen und entsprechend anpassen.

1 Melden Sie sich mithilfe des vSphere Client mit dem Benutzerkonto **platform-operator-00** bei vCenter Server an.

2 Stellen Sie sicher, dass Sie über Lesezugriff auf vSphere-Objekte verfügen, einschließlich Hosts und Cluster, VMs und Vorlagen, Speicher und Netzwerk.

3 Stellen Sie sicher, dass Sie eine Inhaltsbibliothek erstellen und konfigurieren können. Weitere Informationen finden Sie unter Kapitel 5 Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster.

4 Stellen Sie sicher, dass Sie einen vSphere-Namespace erstellen und konfigurieren können. Dazu gehören das Hinzufügen von Benutzern, das Zuordnen der Inhaltsbibliothek und das Zuweisen einer Speicherrichtlinie. Weitere Informationen finden Sie unter Kapitel 6 Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern.

5 Stellen Sie sicher, dass Sie sich mit dem Kubernetes-CLI-Tools für vSphere bei Supervisor anmelden können. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu TKG-Dienst-Clustern mithilfe der vCenter SSO-Authentifizierung.

6 Stellen Sie sicher, dass Sie einen TKG-Cluster auf Supervisor mithilfe von Kubectl bereitstellen können. Weitere Informationen finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl.

7 Stellen Sie sicher, dass Sie sich mithilfe von Kubectl auf Supervisor beim TKG-Cluster anmelden können. Weitere Informationen finden Sie unter Herstellen einer Verbindung mit einem TKG-Dienst-Cluster als vCenter Single Sign-On-Benutzer mit Kubectl.

8 Stellen Sie sicher, dass Sie Arbeitslasten im TKG-Cluster auf Supervisor bereitstellen können. Weitere Informationen finden Sie unter Kapitel 12 Bereitstellen von Arbeitslasten auf TKG-Dienst-Clustern.

9 Stellen Sie sicher, dass Sie verschiedene operative Aufgaben für den TKG-Cluster auf Supervisor durchführen können. Weitere Informationen finden Sie unter Kapitel 8 Betreiben von TKG-Dienstclustern.

10 Stellen Sie mithilfe des vSphere Client sicher, dass Sie den auf Supervisor bereitgestellten TKG-Cluster im vSphere-Namespace anzeigen können.

11 Stellen Sie mithilfe des vSphere Client sicher, dass Sie Supervisor- und TKG-Clusterobjekte überwachen können.

12 Führen Sie einen Negativtest durch, um sicherzustellen, dass Sie bestimmte Aufgaben, die für vSphere Administratoren reserviert sind, nicht ausführen können. Beispielsweise sollten Sie keine neue vSphere-Speicherrichtlinie oder ein vSphere-Netzwerk erstellen können. Sie sollten auch nicht in der Lage sein, das Arbeitslastmanagement zu deaktivieren.

13 Passen Sie die Rollenberechtigungen entsprechend an, um Ihre Sicherheits- und Geschäftsanforderungen zu erfüllen.

Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster

5

Tanzu Kubernetes-Versionen (TKrs) stellen die Kubernetes-Softwareverteilung für TKG-Dienst-Cluster bereit. TKrs werden von VMware als Vorlagen für virtuelle Maschinen verteilt, die Sie mit der Plattform mithilfe einer vCenter-Inhaltsbibliothek synchronisieren.

Lesen Sie als Nächstes die folgenden Themen:

- Verwenden von Kubernetes-Versionen mit TKG-Dienstclustern
- Erforderliche Rollenberechtigungen für die Verwaltung von Inhaltsbibliotheken
- Erstellen einer abonnierten Inhaltsbibliothek
- Erstellen einer lokalen Inhaltsbibliothek (für Air-Gapped-Clusterbereitstellung)
- Aktivieren der Veröffentlichung einer lokalen Inhaltsbibliothek
- Bearbeiten einer vorhandenen Inhaltsbibliothek
- Migrieren einer Inhaltsbibliothek
- Grundlegendes zur TKr-Auflösung

Verwenden von Kubernetes-Versionen mit TKG-Dienstclustern

Eine Tanzu Kubernetes-Version (TKr) bietet die von VMware signierte und unterstützte Kubernetes-Softwareverteilung für die Verwendung mit TKG-Dienstclustern. Das TKr-Format wird für vSphere 8 aktualisiert, um Pakete und mehrere Betriebssysteme zu unterstützen.

Versionshinweise zu TKr

In den Versionshinweisen zu Tanzu Kubernetes-Versionen finden Sie eine vollständige Liste der verfügbaren TKrs, die neuen Funktionen für jede Version, bekannte Probleme und die TKr-Kompatibilität.

TKr – Verteilung und Verbrauch

VMware verteilt Tanzu Kubernetes-Versionen über ein Content Delivery Network. Mithilfe einer vSphere-Inhaltsbibliothek verknüpfen Sie TKrs mit vSphere-Namespaces. Um den Verbrauch von TKrs zu automatisieren, verwenden Sie eine Erstellen einer abonnierten Inhaltsbibliothek. Verwenden Sie für Umgebungen mit eingeschränktem Internetzugriff eine Erstellen einer lokalen Inhaltsbibliothek (für Air-Gapped-Clusterbereitstellung).

Jede Tanzu Kubernetes-Version wird als OVA-Vorlage bereitgestellt. Der TKr-Controller auf Supervisor verwendet die OVA-Vorlage, um die VMs für TKG-Clusterknoten zu erstellen. Die VM-Festplattengröße wird durch die TKr-OVA-Vorlage festgelegt. Sie geben CPU- und RAM-Ressourcen mithilfe von Verwenden von VM-Klassen mit TKG-Dienstclustern an.

Die TKr-Inhaltsbibliothek ist nicht auf den Namespace-Geltungsbereich beschränkt. Alle vSphere-Namespaces verwenden dieselbe TKr-Inhaltsbibliothek für TKG mit Supervisor. Wenn Sie die TKr-Inhaltsbibliothek für einen vSphere-Namespace ändern, wird sie auch für alle anderen aktualisiert.

TKr NAME-Zeichenfolge

Listen Sie mithilfe des Befehls `kubectl get tkr` die TKr-Images auf, die im vSphere-Namespace zur Verfügung stehen. Beispiel:

```
kubectl get tkr
NAME                                    VERSION                                 READY
COMPATIBLE   CREATED
v1.16.14---vmware.1-tkg.1.ada4837       v1.16.14+vmware.1-tkg.1.ada4837         False
False        19d
v1.17.17---vmware.1-tkg.1.d44d45a       v1.17.17+vmware.1-tkg.1.d44d45a         False
False        19d
v1.18.19---vmware.1-tkg.1.17af790       v1.18.19+vmware.1-tkg.1.17af790         False
False        19d
v1.19.16---vmware.1-tkg.1.df910e2       v1.19.16+vmware.1-tkg.1.df910e2         False
False        19d
v1.20.12---vmware.1-tkg.1.b9a42f3       v1.20.12+vmware.1-tkg.1.b9a42f3         False
False        19d
v1.21.6---vmware.1-tkg.1.b3d708a        v1.21.6+vmware.1-tkg.1.b3d708a          True
True         19d
v1.22.9---vmware.1-tkg.1.cc71bc8        v1.22.9+vmware.1-tkg.1.cc71bc8          True
True         19d
v1.23.8---vmware.2-tkg.2-zshippable     v1.23.8+vmware.2-tkg.2-zshippable       True
True         19d
v1.23.8---vmware.3-tkg.1                v1.23.8+vmware.3-tkg.1                  True
True         19d
```

Sie verwenden die TKr NAME-Zeichenfolge, um TKG-Cluster auf Supervisor bereitzustellen. Wenn Sie die TanzuKubernetesCluster v1alpha3-API verwenden, geben Sie die vollständige TKr NAME-Zeichenfolge im Feld `tkr.reference.name` an. Wenn Sie die Cluster-API v1beta1 verwenden, geben Sie die vollständige TKr NAME-Zeichenfolge im Feld `topology.version` an.

Hinweis Verwenden Sie nicht die VERSION-Zeichenfolge, wenn Sie auf die TKr in der Clusterspezifikation verweisen. Das Format muss exakt mit der TKr NAME-Zeichenfolge übereinstimmen.

Der Name der TKr in der Inhaltsbibliothek muss der vollständigen TKr NAME-Zeichenfolge entsprechen. Wenn Sie eine abonnierte Inhaltsbibliothek verwenden, wird die TKr NAME-Zeichenfolge für Sie erstellt. Wenn Sie eine lokale Inhaltsbibliothek verwenden, stellen Sie sicher, dass der für die TKr vergebene Name mit der TKr NAME-Zeichenfolge übereinstimmt. Weitere Informationen finden Sie unter Erstellen einer lokalen Inhaltsbibliothek (für Air-Gapped-Clusterbereitstellung).

TKr-Kompatibilität mit TKG-Dienst

TKrs werden unabhängig vom TKG-Dienst und vom Supervisor freigegeben und aktualisiert.

Zum Bereitstellen eines TKG-Clusters muss die TKr mit dem TKG-Dienst kompatibel sein. Sie können keine TKr verwenden, die nicht mit dem TKG-Dienst kompatibel ist. Darüber hinaus müssen Sie sicherstellen, dass eine kompatible TKr für die zu aktualisierende Version ausgeführt wird.

Sie können die TKr-Kompatibilität mithilfe des Befehls `kubectl get tkr` überprüfen. Die COMPATIBLE-Spalte gibt einen booleschen Wert zurück. `True` bedeutet, dass die TKr kompatibel ist. `False` bedeutet, dass die TKr nicht kompatibel ist.

TKr-Kompatibilität mit vSphere

Das TKr-Format wird für vSphere 8 aktualisiert. TKrs für vSphere 8 können nur auf vSphere 8.x ausgeführt werden. TKrs für vSphere 7.x sind Legacy-Images, die mit vSphere 7 funktionieren. Solche Images können auf vSphere 8 ausgeführt werden, aber nur zu Upgrade-Zwecken. Legacy-TKr-Images werden durch die Anmerkungsbezeichnung `legacy-tkr` angegeben.

Sie können die TKr-Kompatibilität mithilfe der Befehle `kubectl get tkr -o yaml` und `kubectl get tkr --show-labels` überprüfen. Wenn die Anmerkungsbezeichnung `legacy-tkr` vorhanden ist, unterstützt die TKr vSphere 8-Funktionen nicht und sollte nur für das Upgrade von vSphere 7 auf vSphere 8 verwendet werden.

Der folgende Befehl zeigt beispielsweise an, dass es sich bei dem angegebenen Image um die Bezeichnung `legacy-tkr` handelt.

```
kubectl get tkr v1.23.8---vmware.3-tkg.1 -o yaml
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesRelease
metadata:
  creationTimestamp: "2023-03-15T20:33:17Z"
```

```
  finalizers:
  - tanzukubernetesrelease.run.tanzu.vmware.com
  generation: 1
  labels:
    os-arch: amd64
    os-name: photon
    os-type: linux
    os-version: "3.0"
    run.tanzu.vmware.com/legacy-tkr: ""
    v1: ""
    v1.23: ""
    v1.23.8: ""
    v1.23.8---vmware: ""
    v1.23.8---vmware.3: ""
    v1.23.8---vmware.3-tkg: ""
    v1.23.8---vmware.3-tkg.1: ""
  name: v1.23.8---vmware.3-tkg.1
```

Im folgenden Beispiel wird das `--show-labels`-Flag verwendet, um die TKr-Kompatibilität zu überprüfen. Die `legacy-tkr`-Bezeichnung ist vorhanden, sodass das Image nur zum Erstellen eines TKG-Clusters verwendet werden kann.

```
kubectl get tkr v1.23.8---vmware.3-tkg.1 --show-labels
NAME                        VERSION                  READY   COMPATIBLE   CREATED   LABELS
v1.23.8---vmware.3-tkg.1    v1.23.8+vmware.3-tkg.1   True    True         19d       os-
arch=amd64,os-name=photon,os-type=linux,os-version=3.0,run.tanzu.vmware.com/legacy-tkr=,
```

Das folgende Beispiel zeigt eine für vSphere 8.x vorgesehene TKr, da die Bezeichnung `legacy-tkr` in der Liste der Bezeichnungen nicht vorhanden ist.

```
kubectl get tkr v1.28.8---vmware.1-fips.1-tkg.2 --show-labels
NAME                                VERSION                          READY   COMPATIBLE
CREATED   LABELS
v1.28.8---vmware.1-fips.1-tkg.2   v1.28.8+vmware.1-fips.1-tkg.2   True    True
21d       os-arch=amd64,os-name=photon,os-type=linux,os-version=5.0,tkr.tanzu.vmware.com/
standard=,v1.28.8---vmware.1-fips.1-tkg.2=,v1.28.8---vmware.1-fips.1-tkg=,v1.28.8---vmware.1-
fips.1=,v1.28.8---vmware.1-fips=,v1.28.8---vmware.1=,v1.28.8---vmware=,v1.28.8=,v1.28=,v1=
```

Wenn Sie den entsprechenden Befehl mit `-o yaml` ausführen, wird auch angezeigt, dass die Bezeichnung `legacy-tkr` nicht vorhanden ist. Dies spricht dafür, dass die TKr speziell für vSphere 8.x entwickelt wurde.

```
 kubectl get tkr v1.28.8---vmware.1-fips.1-tkg.2 -o yaml
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesRelease
metadata:
  creationTimestamp: "2024-05-08T20:03:57Z"
  finalizers:
  - tanzukubernetesrelease.run.tanzu.vmware.com
  generation: 2
  labels:
    os-arch: amd64
    os-name: photon
```

```
    os-type: linux
    os-version: "5.0"
    tkr.tanzu.vmware.com/standard: ""
    v1: ""
    v1.28: ""
    v1.28.8: ""
    v1.28.8---vmware: ""
    v1.28.8---vmware.1: ""
    v1.28.8---vmware.1-fips: ""
    v1.28.8---vmware.1-fips.1: ""
    v1.28.8---vmware.1-fips.1-tkg: ""
    v1.28.8---vmware.1-fips.1-tkg.2: ""
  name: v1.28.8---vmware.1-fips.1-tkg.2
...
```

TKr-Betriebssystem-Image-Format

Das TKr-Betriebssystem-Image-Format unterstützt mehrere Betriebssystem-Images für eine einzelne TKr. Dies bedeutet, dass es eine einzige Tanzu Kubernetes-Version für eine bestimmte Kubernetes-Version für alle unterstützten Betriebssysteme gibt. Dies sind derzeit Photon OS und Ubuntu. Photon OS ist das standardmäßige Betriebssystem-Image-Format.

Standardmäßig wird die Photon OS-Edition der benannten TKr für TKG-Clusterknoten verwendet. Wenn die referenzierte TKr das Betriebssystem-Image-Format unterstützt und eine Ubuntu-Betriebssystemedition verfügbar ist, geben Sie mithilfe der Anmerkung `run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu` die Ubuntu-Betriebssystemedition der TKr an. Beispiel: Die folgende Kapitel 7 Bereitstellen von TKG-Dienstclustern verwendet die Ubuntu-Edition der benannten TKr.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkgs-cluster-ubuntu
  namespace: tkgs-cluster-ns
  annotations:
    run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
spec:
   topology:
     controlPlane:
       ...
       tkr:
         reference:
           name: v1.28.8---vmware.1-fips.1-tkg.2
```

Das Betriebssystem-Image-Format unterstützt heterogene Clusterbereitstellungen. Beispielsweise erstellt das folgende Clustermanifest einen Tanzu Kubernetes-Cluster mit dem Standardbetriebssystem Photon OS für die Steuerungsebenenknoten und Ubuntu für die Worker-Knoten. Die TKr-Version wird im Abschnitt der Steuerungsebene referenziert, und die Anmerkung gibt Ubuntu für den benannten Worker-Knotenpool an.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkgs-cluster-multiOS
  namespace: tkgs-cluster-ubuntu
  annotations:
    //Worker nodes annotation
    run.tanzu.vmware.com/resolve-os-image.np-1: os-name=ubuntu
spec:
  topology:
    controlPlane:
      tkr:
        reference:
          name: v1.28.8---vmware.1-fips.1-tkg.2
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkgs-storage-profile
    nodePools:
    - replicas: 3
      name: np-1
      vmClass: guaranteed-medium
      storageClass: tkgs-storage-profile
```

Wenn das System auf vSphere 8 aktualisiert wird, werden vorhandene TKrs automatisch in das TKr-Betriebssystem-Image-Format mit Referenz auf ein einzelnes Betriebssystem-Image konvertiert. Dadurch können Legacy-TKrs für das Upgrade auf ein Nicht-Legacy-TKr kompatibel sein.

Legacy-TKrs haben zwei Editionen: Photon und Ubuntu. Wenn die Legacy-TKr über eine Ubuntu-spezifische Edition verfügt, können Sie die vollständige Versionszeichenfolge (mit „ubuntu“) verwenden und die Anmerkungsbezeichnung weglassen oder die kurze Versionszeichenfolge (ohne „ubuntu“) verwenden und die Versionsbezeichnung einschließen.

TKr-Pakete

TKr-Images, die mit vSphere 8 kompatibel sind, werden auf ein paketbasiertes Framework für zentrale Komponenten aktualisiert, etwa das Container Storage Interface (CSI) und das Container Network Interface (CNI). Wenn Sie die v1beta-API verwenden, werden diese Komponenten mithilfe benutzerdefinierter Ressourcendefinitionen geändert oder aktualisiert.

Um die Pakete anzuzeigen, aus denen sich eine TKr zusammensetzt, führen Sie den folgenden Befehl aus:

```
kubectl get tkr TKR-NAME -o yaml
```

Beispiel:

```
kubectl get tkr v1.28.8---vmware.1-fips.1-tkg.2 -o yaml
```

Der Befehl gibt alle Pakete in der TKr zurück. Beispiel:

```
spec:
  bootstrapPackages:
  - name: antrea.tanzu.vmware.com.1.13.3+vmware.3-tkg.2-vmware
  - name: vsphere-pv-csi.tanzu.vmware.com.3.1.0+vmware.1-tkg.6-vmware
  - name: vsphere-cpi.tanzu.vmware.com.1.28.0+vmware.1-tkg.1-vmware
  - name: kapp-controller.tanzu.vmware.com.0.48.2+vmware.1-tkg.1-vmware
  - name: guest-cluster-auth-service.tanzu.vmware.com.1.3.3+tkg.1-vmware
  - name: metrics-server.tanzu.vmware.com.0.6.2+vmware.3-tkg.5-vmware
  - name: secretgen-controller.tanzu.vmware.com.0.15.0+vmware.1-tkg.1-vmware
  - name: pinniped.tanzu.vmware.com.0.25.0+vmware.2-tkg.1-vmware
  - name: capabilities.tanzu.vmware.com.0.32.1+vmware.1
  - name: gateway-api.tanzu.vmware.com.1.0.0+vmware.1-tkg.1-vmware
  - name: calico.tanzu.vmware.com.3.26.3+vmware.1-tkg.1-vmware
```

Unter v1beta1-Beispiel: Cluster mit Calico-CNI finden Sie ein Beispiel für einen Anwendungsfall.

Migrieren von TKr-Betriebssystemtypen

Direkte Cluster-Updates zwischen TKr-Betriebssystemen werden nicht unterstützt. Dies bedeutet beispielsweise, dass Sie kein Upgrade eines TKG-Clusters mit TKr v1.27.11 Photon auf TKr v1.28.8 Ubuntu durchführen können.

Wenn Sie den von einem TKG-Cluster verwendeten TKr-Betriebssystemtyp ändern möchten, verwenden Sie folgendes Verfahren. In diesem Beispiel verwendet der ursprüngliche Cluster TKr-Photon, während auf dem Zielcluster TKr Ubuntu ausgeführt wird.

- Sichern Sie die Photon-basierten TKG-Clusterarbeitslasten mithilfe von Velero.

  Weitere Informationen hierzu finden Sie unter Kapitel 20 Sichern und Wiederherstellen von TKG-Dienstclustern und -Arbeitslasten.

- Stellen Sie einen neuen TKG-Cluster mithilfe der Ubuntu-TKr bereit.

  Weitere Informationen hierzu finden Sie unter Kapitel 7 Bereitstellen von TKG-Dienstclustern.

- Stellen Sie die TKG-Clusterarbeitslasten mithilfe von Velero im Ubuntu-Cluster wieder her.

  Weitere Informationen hierzu finden Sie unter Kapitel 20 Sichern und Wiederherstellen von TKG-Dienstclustern und -Arbeitslasten.

TKr-Härtung

Security Technical Implementation Guides (STIG) für Systemkomponenten, einschließlich Supervisor und TKrs, stehen zur Verfügung. Weitere Informationen finden Sie unter Tanzu STIG-Härtung.

Erstellen einer eigenen TKr

Ab TKr v1.25.7 für vSphere 8.x können Sie benutzerdefinierte TKr-Maschinen-Images für TKG-Cluster auf vSphere 8 erstellen. Ein benutzerdefiniertes Maschinen-Image enthält ein unterstütztes Betriebssystem in einer unterstützten Version, eine Kubernetes-Version basierend auf einer freigegebenen TKr und alle von Ihnen vorgenommenen Anpassungen.

Zum Erstellen benutzerdefinierter Maschinen-Images für TKG-Clusterknoten verwenden Sie den vSphere Tanzu Kubernetes Grid Image Builder. Weitere Informationen zum Erstellen benutzerdefinierter Images, unterstützter TKr-Versionen und unterstützter Anpassungen finden Sie in der Dokumentation.

Erforderliche Rollenberechtigungen für die Verwaltung von Inhaltsbibliotheken

Für die Verwaltung von Inhaltsbibliotheken sind mehrere vSphere Rollenberechtigungen erforderlich. Wenn Sie die Rolle „vSphere Administrator“ nicht verwenden, lesen Sie nach Bedarf die Liste der Berechtigungen.

Erforderliche Berechtigungen zum Verwalten von Inhaltsbibliotheken

Für die Verwaltung der Inhaltsbibliothek sind die folgenden vSphere Rollenberechtigungen erforderlich.

Diese Berechtigungen sind in der Rolle vSphere-Administrator enthalten.

Informationen zum Erstellen einer dedizierten Rolle, die diese Berechtigungen enthält, finden Sie unter Erstellen einer dedizierten Gruppe und Rolle für Plattformoperatoren.

```
- Content Library
  - Add library item &
  - Check in a template &
  - Check out a template &
  - Create local library &
  - Create subscribed library &
  - Delete library item &
  - Delete local library &
  - Delete subscribed library &
  - Download files &
  - Evict library item &
  - Evict subscribed library &
  - Import storage &
  - Probe subscription information &
  - Read storage &
  - Sync library item &
  - Sync subscribed library &
  - Type introspection &
  - Update configuration settings &
  - Update library &
```

```
- Update library item &
- Update local library &
- Update subscribed library &
- View configuration settings &
```

Erstellen einer abonnierten Inhaltsbibliothek

Um TKG-Cluster auf Supervisor bereitzustellen, können Sie eine abonnierte Inhaltsbibliothek erstellen und Tanzu Kubernetes-Versionen synchronisieren. Mit einer abonnierten Inhaltsbibliothek können Sie die Verteilung von TKRs automatisieren und mit den aktuellen Versionen auf dem neuesten Stand bleiben.

Für TKG auf Supervisor veröffentlicht VMware Tanzu Kubernetes-Versionen in einem Content Delivery Network. Sie können eine Inhaltsbibliothek erstellen, die diese Image-Publikationen abonniert. Sie können den Synchronisierungsmodus auswählen: „sofort“ oder „bei Bedarf“.

Warnung Die sofortige Synchronisierung von Tanzu Kubernetes-Versionen in einem öffentlichen Content Delivery Network kann viel Zeit und Festplattenspeicher in Anspruch nehmen.

Voraussetzungen

Die Inhaltsbibliothekfunktion ist ein Merkmal von vCenter Server, auf das sich TKG auf Supervisor stützt. Weitere Informationen finden Sie unter „Verwenden von Inhaltsbibliotheken“ in der Dokumentation zur Verwaltung virtueller vSphere-Maschinen.

Verfahren

1 Melden Sie sich über vSphere Client bei vCenter Server an.

2 Wählen Sie **Menü > Inhaltsbibliotheken** aus.

3 Klicken Sie auf **Erstellen**.

Der Assistent **Neue Inhaltsbibliothek** wird geöffnet.

4 Geben Sie **Name und Speicherort** der Inhaltsbibliothek an und klicken Sie auf **Weiter**, wenn Sie fertig sind.

Bereich	Beschreibung
Name	Geben Sie einen beschreibenden Namen ein, z. B. **`TKR-sub`**.
Anmerkungen	Geben Sie eine Beschreibung an, z. B **`Abonnementbibliothek für TKRs für TKG2`**.
vCenter Server	Wählen Sie die vCenter Server-Instanz aus, in der die **Arbeitslastverwaltung** aktiviert ist.

5 Konfigurieren Sie das Abonnement der Inhaltsbibliothek auf der Seite **Inhaltsbibliothek konfigurieren** und klicken Sie auf **Weiter**, wenn Sie fertig sind.

   a Wählen Sie die Option **Abonnierte Inhaltsbibliothek** aus.

   b Geben Sie die Adresse der **Abonnement-URL** des Herausgebers ein.

   ```
   https://wp-content.vmware.com/v2/latest/lib.json
   ```

   c Wählen Sie für die Option **Inhalt herunterladen** eine der folgenden Optionen aus:

Option	Beschreibung
Sofort	Der Abonnementvorgang synchronisiert die Metadaten und Images der Bibliothek. Wenn Elemente aus der veröffentlichten Bibliothek gelöscht werden, bleibt deren Inhalt am Speicherort der abonnierten Bibliothek erhalten, und Sie müssen ihn manuell löschen.
Bei Bedarf	Der Abonnementvorgang synchronisiert nur die Metadaten der Bibliothek. Der Tanzu Kubernetes Grid Service lädt die Images bei der Veröffentlichung herunter. Wenn Sie das Element nicht mehr benötigen, können Sie den Inhalt des Elements löschen, um Speicherplatz freizugeben. Um Speicherplatz zu sparen, wird diese Option empfohlen.

6 Akzeptieren Sie den Fingerabdruck des SSL-Zertifikats, wenn Sie dazu aufgefordert werden.

   Der Fingerabdruck des SSL-Zertifikats wird auf Ihrem System gespeichert, bis Sie die abonnierte Inhaltsbibliothek aus dem Bestand löschen.

7 Konfigurieren Sie die OVF-Sicherheitsrichtlinie auf der Seite **Sicherheitsrichtlinie anwenden** und klicken Sie abschließend auf **Weiter**.

   a Klicken Sie auf **Sicherheitsrichtlinie anwenden**.

   b Klicken Sie auf **OVF-Standardrichtlinie**.

   Wenn Sie diese Option auswählen, überprüft das System während des Synchronisierungsvorgangs das OVF-Signaturzertifikat. Eine OVF-Vorlage, die die Zertifikatvalidierung nicht besteht, wird mit dem Tag **Überprüfung fehlgeschlagen** gekennzeichnet. Die Metadaten der Vorlage werden beibehalten, aber die OVF-Dateien können nicht synchronisiert werden.

   Hinweis Derzeit ist die **OVF-Standardrichtlinie** die einzige unterstützte Sicherheitsrichtlinie.

8 Wählen Sie auf der Seite **Speicher hinzufügen** einen Datenspeicher als Speicherort für die Inhalte der Inhaltsbibliothek aus und klicken Sie auf **Weiter**.

9 Überprüfen Sie auf der Seite **Bereit zum Abschließen** die Details und klicken Sie auf **Beenden**.

10 Wählen Sie auf der Seite **Inhaltsbibliotheken** die neue Inhaltsbibliothek aus, die Sie erstellt haben.

11 Bestätigen oder schließen Sie die Synchronisierung der Bibliotheksinhalte ab.

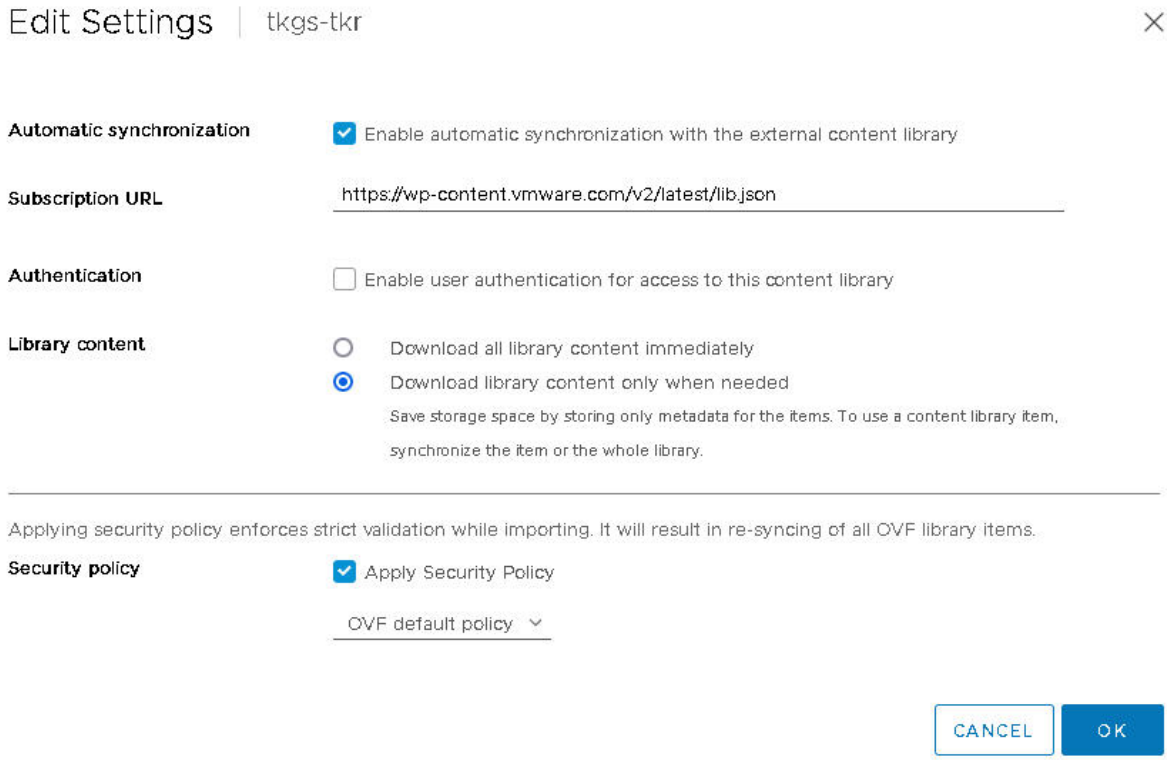
Synchronisierungsoption	Beschreibung
Sofort	Wenn Sie den gesamten Inhalt sofort herunterladen möchten, bestätigen Sie, dass die Bibliothek synchronisiert ist. Um die synchronisierten Bibliotheksinhalte anzuzeigen, klicken Sie auf **Vorlagen > OVF- und OVA-Vorlagen**.
Bei Bedarf	Wenn Sie die Bibliothek bei Bedarf synchronisieren möchten, stehen Ihnen zwei Optionen zur Verfügung: ■ Verwenden Sie **Aktionen** > **Synchronisieren**, um die gesamte Bibliothek zu synchronisieren ■ Klicken Sie mit der rechten Maustaste auf ein Element und wählen Sie **Synchronisieren** aus, um es nur zu synchronisieren. Um die synchronisierten Bibliotheksinhalte anzuzeigen, klicken Sie auf **Vorlagen > OVF- und OVA-Vorlagen**.

12 Wenn Sie die Option **Bei Bedarf** ausgewählt haben, laden Sie die zu verwendenden OVF-Vorlagen herunter.

Wenn Sie die Option **Bei Bedarf** ausgewählt haben, sehen Sie, dass nicht die Image-Dateien, sondern nur die Metadateien lokal gespeichert werden. Um die Vorlagendateien herunterzuladen, wählen Sie das Element aus, klicken Sie mit der rechten Maustaste und wählen Sie **Element synchronisieren** aus.

13 Um die Einstellungen der abonnierten Inhaltsbibliothek zu aktualisieren, klicken Sie auf **Aktionen > Einstellungen bearbeiten**.

Einstellung	Wert
URL für Abonnement	`https://wp-content.vmware.com/v2/latest/lib.json`
Authentifizierung	Nicht aktiviert
Bibliotheksinhalt	Bei Bedarf herunterladen
Sicherheitsrichtlinie	OVF-Standardrichtlinie



Nächste Schritte

Die Tanzu Kubernetes Release-Inhaltsbibliothek muss mit jedem vSphere-Namespace verknüpft sein, in dem Sie TKG-Cluster bereitstellen. Weitere Informationen hierzu finden Sie unter Kapitel 6 Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern.

Erstellen einer lokalen Inhaltsbibliothek (für Air-Gapped-Clusterbereitstellung)

Um TKG-Cluster auf Supervisor bereitzustellen, können Sie eine lokale Inhaltsbibliothek erstellen und Tanzu Kubernetes-Versionen importieren. Der typische Anwendungsfall für eine lokale Inhaltsbibliothek sind Umgebungen mit Internetbeschränkung (air-gapped).

Das Erstellen einer lokalen Inhaltsbibliothek umfasst das Konfigurieren der Bibliothek, das Herunterladen der OVA-Dateien und das Importieren in die lokale Inhaltsbibliothek.

Voraussetzungen

Die Inhaltsbibliothekfunktion ist ein Merkmal von vCenter Server, auf das sich TKG auf Supervisor stützt. Weitere Informationen finden Sie unter Verwenden von Inhaltsbibliotheken .

Verfahren

1 Melden Sie sich über vSphere Client bei vCenter Server an.

2 Klicken Sie auf **Menü**.

3 Klicken Sie auf **Inhaltsbibliothek**.

4 Klicken Sie auf **Erstellen**.

Das System zeigt den Assistenten für **Neue Inhaltsbibliothek** an.

5 Geben Sie **Name und Speicherort** der Inhaltsbibliothek an und klicken Sie auf **Weiter**, wenn Sie fertig sind.

Bereich	Beschreibung
Name	Geben Sie einen aussagekräftigen Namen ein, wie z. B. `TKr-local`.
Anmerkungen	Geben Sie eine Beschreibung an, z. B `Lokale Bibliothek für TKrs für TKG`.
vCenter Server	Wählen Sie die vCenter Server-Instanz aus, auf der vSphere IaaS control plane aktiviert ist.

6 Wählen Sie auf der Seite **Inhaltsbibliothek konfigurieren** die Option **Lokale Inhaltsbibliothek** aus und klicken Sie auf **Weiter**.

Für lokale Inhaltsbibliotheken importieren Sie die OVF-Vorlagen, die Sie verwenden möchten, manuell.

7 Konfigurieren Sie die OVF-Sicherheitsrichtlinie auf der Seite **Sicherheitsrichtlinie anwenden** und klicken Sie abschließend auf **Weiter**.

a Klicken Sie auf **Sicherheitsrichtlinie anwenden**.

b Klicken Sie auf **OVF-Standardrichtlinie**.

Wenn Sie diese Option auswählen, überprüft das System während des Synchronisierungsvorgangs das OVF-Signaturzertifikat. Eine OVF-Vorlage, die die Zertifikatvalidierung nicht besteht, wird mit dem Tag **Überprüfung fehlgeschlagen** gekennzeichnet. Die Metadaten der Vorlage werden beibehalten, aber die OVF-Dateien können nicht synchronisiert werden.

Hinweis Derzeit ist die **OVF-Standardrichtlinie** die einzige unterstützte Sicherheitsrichtlinie.

8 Wählen Sie auf der Seite **Speicher hinzufügen** einen Datenspeicher als Speicherort für die Inhalte der Inhaltsbibliothek aus und klicken Sie auf **Weiter**.

9 Überprüfen Sie auf der Seite **Bereit zum Abschließen** die Details und klicken Sie auf **Beenden**.

10 Wählen Sie auf der Seite **Inhaltsbibliotheken** die neue Inhaltsbibliothek aus, die Sie erstellt haben.

11 Laden Sie die OVA-Dateien für jede Tanzu Kubernetes-Version herunter, die Sie in die lokale Inhaltsbibliothek importieren möchten.

a Navigieren Sie in einem Browser zur folgenden URL:

https://wp-content.vmware.com/v2/latest/

b Klicken Sie auf das Verzeichnis für das gewünschte Image. In der Regel ist dieses Verzeichnis die neueste oder aktuellste Version der Kubernetes-Distribution.

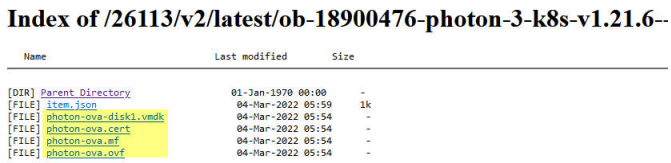
Beispiel:

```
ob-18186591-photon-3-k8s-v1.20.7---vmware.1-tkg.1.7fb9067
```

Wichtig Sie müssen den Distributionsnamen verwenden, um die Dateien in die lokale Inhaltsbibliothek zu importieren. Kopieren Sie den Zielnamen in eine Datei oder lassen Sie den Browser geöffnet, bis Sie den Vorgang abgeschlossen haben. Der erforderliche Teil der Namenszeichenfolge, den Sie basierend auf dem obigen Beispiel benötigen, lautet `photon-3-k8s-v1.20.7---vmware.1-tkg.1.7fb9067`.

c Klicken Sie für jede der folgenden Dateien mit der rechten Maustaste und wählen Sie **Link speichern unter** aus.

- `photon-ova-disk1.vmdk`
- `photon-ova.cert`
- `photon-ova.mf`
- `photon-ova.ovf`



d Stellen Sie sicher, dass jede Datei erfolgreich in Ihr lokales Dateisystem heruntergeladen wurde.

Hinweis Die importierten Dateien sind die OVF- und VMDK-Dateien. Wenn jedoch eine Sicherheitsrichtlinie angewendet wird, müssen alle vier Dateien, einschließlich des Zertifikats (`*.cert`) und des Manifests (`*.mf`), während des Imports im Quellverzeichnis vorhanden sein. Wenn die Zertifikats- und Manifestdateien während des Imports nicht verfügbar sind, kann das importierte Tanzu Kubernetes Release nicht verwendet werden.

12 Importieren Sie die OVA- und VMDK-Dateien in die lokale Inhaltsbibliothek.

a Wählen Sie **Menü > Inhaltsbibliotheken >** aus.

b Klicken Sie in der Liste der **Inhaltsbibliotheken** auf den Link für den Namen der von Ihnen erstellten lokalen Inhaltsbibliothek.

c Klicken Sie auf **Aktionen**.

d Wählen Sie **Element importieren** aus.

e Wählen Sie im Fenster **Bibliothekselement importieren** die Option **Lokale Datei** aus.

f Klicken Sie auf **Dateien hochladen**.

g Wählen Sie die Dateien `photon-ova.ovf` und `photon-ova-disk1.vmdk` aus.

Folgende Meldung wird angezeigt: `2 files ready to import`. Jede Datei wird mit einem grünen Häkchen neben Ihrem Namen aufgelistet.

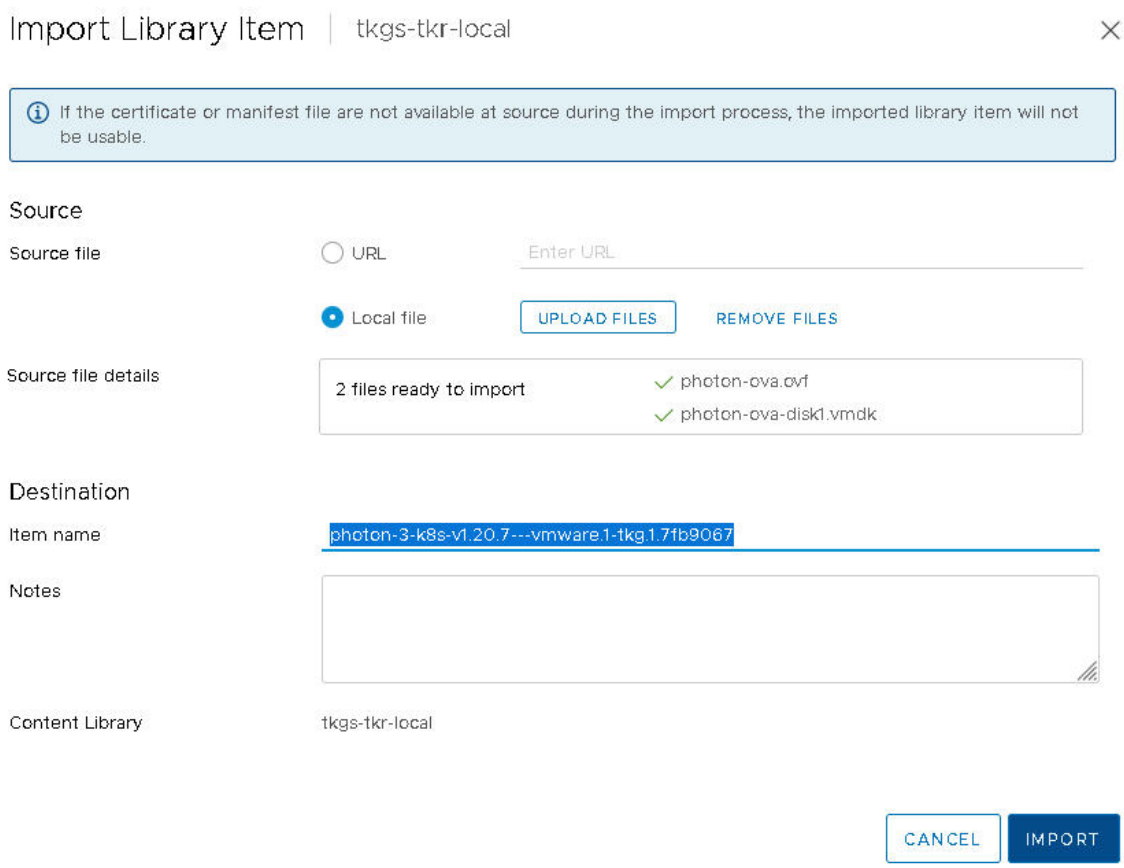
h Ändern Sie den Namen für das **Zielelement** so, dass er die Betriebssystem-Image-Version plus die Kubernetes-Version aus dem Verzeichnis, in das Sie die Dateien heruntergeladen haben, wiedergibt.

Beispiel:

```
photon-3-k8s-v1.20.7---vmware.1-tkg.1.7fb9067
```

Warnung Der unter **Zielelement** für die Inhaltsbibliothek angegebene Name muss der Ordnernamenzeichenfolge für das gewünschte Tanzu Kubernetes Release genau entsprechen. Wenn die Namen nicht übereinstimmen, kann Supervisor das Image nicht in ein gültiges Tanzu Kubernetes Release auflösen.

i Klicken Sie auf **Import**.



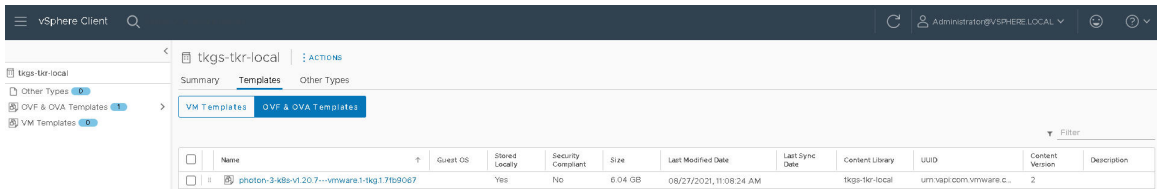
13 Stellen Sie sicher, dass die lokale Inhaltsbibliothek mit der Tanzu Kubernetes-Version aufgefüllt wurde.

a Zeigen Sie den Bereich **Aktuelle Aufgaben** am unteren Rand der Seite an.

b Überwachen Sie die Aufgabe **Inhalt eines Bibliothekselements abrufen** und stellen Sie sicher, dass sie erfolgreich **abgeschlossen** wurde.

c Klicken Sie in der lokalen Inhaltsbibliothek auf **Vorlagen > OVF- und OVA-Vorlagen**.

d Stellen Sie sicher, dass die Tanzu Kubernetes-Version-Metadaten aufgelistet sind und dass ihr Inhalt lokal gespeichert wird.



Nächste Schritte

Die Tanzu Kubernetes-Version-Inhaltsbibliothek muss mit jedem vSphere-Namespace verknüpft sein, in dem Sie TKG-Cluster bereitstellen. Weitere Informationen hierzu finden Sie unter Kapitel 6 Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern.

Aktivieren der Veröffentlichung einer lokalen Inhaltsbibliothek

Sie können eine lokale Inhaltsbibliothek erstellen und die Veröffentlichung aktivieren, sodass andere Inhaltsbibliotheken sie abonnieren können. Mit diesem Verfahren können Sie die Tanzu Kubernetes-Version-Images steuern, die für die Verwendung verfügbar sind, während Sie TKG-Cluster-Operatoren mit einem TKR-Dienst im Abonnementformat bereitstellen.

Anstatt das VMware Content Delivery Network für Tanzu Kubernetes-Versionen zu abonnieren, in dem alle Images veröffentlicht werden und verbleiben, können Sie eine lokale Inhaltsbibliothek erstellen, die eine begrenzte Auswahl an Tanzu Kubernetes-Versionen bereitstellt. Ein solcher Ansatz ist praktisch, wenn Sie die Verfügbarkeit von Images steuern möchten, wie z. B. in kontinuierlichen Build-Umgebungen.

Verfahren

1 Erstellen Sie eine Erstellen einer lokalen Inhaltsbibliothek (für Air-Gapped-Clusterbereitstellung) oder bearbeiten Sie die Einstellungen für eine vorhandene Inhaltsbibliothek.

2 Aktivieren Sie das Kontrollkästchen **Veröffentlichung aktivieren**.

Sie erhalten eine Abonnement-URL für diese lokale Inhaltsbibliothek.

3 Erstellen Sie eine oder mehrere Erstellen einer abonnierten Inhaltsbibliothek mithilfe der Abonnement-URL.

Ergebnisse

Wenn Sie Tanzu Kubernetes-Versionen in die lokale Inhaltsbibliothek mit aktivierter Veröffentlichungsfunktion hochladen, erhalten alle abonnierten Inhaltsbibliotheken die Images bei der Synchronisierung. In der Dokumentation zur Erstellen einer lokalen Inhaltsbibliothek (für Air-Gapped-Clusterbereitstellung) finden Sie Anweisungen zum Hochladen von Images und in

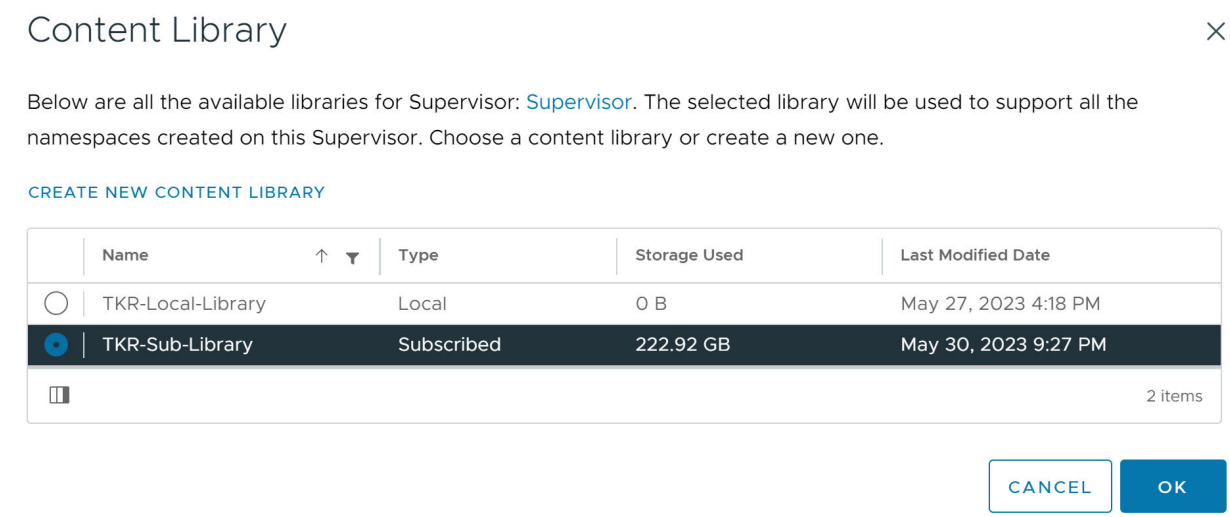
der Dokumentation zu Erstellen einer abonnierten Inhaltsbibliothek finden Sie Informationen zur Erstellung einer abonnierten Inhaltsbibliothek.

Bearbeiten einer vorhandenen Inhaltsbibliothek

Die TKR-Inhaltsbibliothek, die Sie für TKG-Cluster verwenden, ist nicht auf einzelne vSphere-Namespaces beschränkt. Dieselbe TKR-Inhaltsbibliothek wird für alle vSphere-Namespaces auf einer Supervisor-Instanz verwendet. Aus diesem Grund ist das Ändern der TKR-Inhaltsbibliothek nur auf der Supervisor-Ebene zulässig.

Führen Sie diese Schritte aus, um eine vorhandene TKR-Inhaltsbibliothek für die Verwendung durch vSphere-Namespaces zu ändern.

Abbildung 5-1. Bearbeiten der TKR-Inhaltsbibliothek für TKG



Verfahren

1. Melden Sie sich über vSphere Client bei vCenter Server an.
2. Wählen Sie **Arbeitslastverwaltung** aus.
3. Wählen Sie die Supervisor-Instanz aus.
4. Wählen Sie **Supervisor > Allgemein** aus.
5. Wählen Sie Tanzu Kubernetes Grid Service aus.
6. Wählen Sie **Inhaltsbibliothek > BEARBEITEN** aus.
7. Wählen Sie die **Inhaltsbibliothek** aus, die Sie verwenden möchten, und klicken Sie dann auf **OK**.
8. Um eine neue Inhaltsbibliothek zu erstellen, wählen Sie **Neue Inhaltsbibliothek erstellen** aus. Weitere Informationen finden Sie unter Erstellen einer abonnierten Inhaltsbibliothek und Erstellen einer lokalen Inhaltsbibliothek (für Air-Gapped-Clusterbereitstellung).

Ergebnisse

Die ausgewählte Inhaltsbibliothek wird für den Tanzu Kubernetes Grid-Dienst für alle vSphere-Namespaces verwendet, die auf Supervisor erstellt wurden.

Migrieren einer Inhaltsbibliothek

Wenn die TKr-Inhaltsbibliothek die Kapazität erreicht, können Sie zu einer neuen Inhaltsbibliothek mit mehr Speicherkapazität migrieren.

Wenn der vSphere-Administrator eine Inhaltsbibliothek erstellt, gibt der Administrator einen Datenspeicher zum Speichern von Bibliotheksinhalten an, in diesem Fall OVA-Dateien. Wenn im Laufe der Zeit mehr Kubernetes-Versionen verteilt werden, nimmt die Größe der Inhaltsbibliothek zu, da für jedes Update OVA-Dateien hinzugefügt werden. Obwohl die Inhaltsbibliothek keine explizite Kapazität aufweist, ist ihre Datenspeicherkapazität begrenzt.

Wenn die Kapazitätsgrenze der Inhaltsbibliothek erreicht ist, wird möglicherweise die Meldung `Internal error occurred: get library items failed for.` angezeigt. In diesem Fall können Sie die TKG-Cluster zu einer neuen Inhaltsbibliothek migrieren, um die Speicherkapazität zu erhöhen. Die Migration erfolgt mithilfe des vSphere Client.

Verfahren

1. Erstellen Sie eine neue Inhaltsbibliothek mit ausreichender Kapazität für den Zielcluster.
2. Melden Sie sich über vSphere Client bei vCenter Server an.
3. Wählen Sie **Menü > Hosts und Cluster** aus.
4. Wählen Sie das vSphere-Clusterobjekt aus, in dem der Supervisor mit dem Tanzu Kubernetes-Cluster bereitgestellt wird.
5. Wählen Sie die Registerkarte **Konfigurieren** aus.
6. Wählen Sie im Navigationsbereich die Option **Namespaces > Allgemein >** aus.
7. Klicken Sie im Hauptbereich neben dem Abschnitt **Inhaltsbibliothek** auf **Bearbeiten**.
8. Wählen Sie die neue Inhaltsbibliothek aus, die Sie erstellt haben, und klicken Sie auf **OK**.

   Durch diese Aktion wird das Update der Clusterkonfiguration ausgelöst.

   Hinweis Nachdem Sie die Inhaltsbibliothek geändert haben, dauert es unter Umständen bis zu 10 Minuten, bis der TKG-Cluster die Änderung aus der Inhaltsquelle übernimmt.

Grundlegendes zur TKr-Auflösung

In diesem Thema wird die Auflösung von TKr-Images durch das System beschrieben.

TKr-Auflösung

Wenn das Clusterobjekt erstellt oder aktualisiert wird, ruft der Kubernetes-API-Server den Webhook zur Änderung des TKr Resolvers auf. Der Cluster (oder seine ClusterClass) muss über die Anmerkung `run.tanzu.vmware.com/resolve-tkr` verfügen. Ansonsten wird die TKr-Auflösung vollständig übersprungen. Der TKr Resolver verwendet den Wert der Anmerkung `run.tanzu.vmware.com/resolve-tkr` als Bezeichnungsabfrage, um die Gruppe der potenziellen TKrs einzuschränken. Mit einer leeren Zeichenfolge werden alle TKrs ausgewählt.

Die Werte der Anmerkung `run.tanzu.vmware.com/resolve-os-image` in der Clustertopologie „controlPlain“ und „machineDeployments“ werden als Bezeichnungsselektoren für OSImage-Objekte eingesetzt, die jeweils mit „controlPlain“ und „machineDeployments“ verwendet werden sollen. Genau ein OSImage, das von der aufgelösten TKr zur Verfügung gestellt wird, muss die Abfrage für „controlPlane“ oder eine der „machineDeployments“ erfüllen.

Die angegebene spec.topology.version des Clusters wird als Versionspräfix verwendet. Der TKr Resolver-Webhook sucht nach der neuesten verfügbaren TKR, die die oben genannten Einschränkungen erfüllt. Wenn diese nicht gefunden wird, wird die Anforderung zum Erstellen/Aktualisieren des Clusters abgelehnt.

Der TKR Resolver-Webhook ändert den Cluster:

- Die Bezeichnung „run.tanzu.vmware.com/tkr“ ist auf den Namen der aufgelösten TKR festgelegt
- Die spec.topology.version des Clusters ist auf die Kubernetes-Version der aufgelösten TKR festgelegt
- Die Clustervariable TKR_DATA wurde aktualisiert und enthält nun eine Zuordnung zwischen der Kubernetes-Version und der Wertemenge aus der TKR und dem OSImage für die controlPlane.
- Überschreibungen der Variable TKR_DATA für einzelne machineDeployments werden so aktualisiert, dass sie eine Zuordnung zwischen der Kubernetes-Version und der Wertemenge aus der TKR und dem OSImage für die machineDeployment enthalten.

Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern

6

Ein vSphere-Namespace stellt die Laufzeitumgebung für TKG-Dienst-Cluster bereit. Um einen TKG-Dienst-Cluster bereitzustellen, konfigurieren Sie zunächst einen vSphere-Namespace mit Benutzern, Rollen, Computing, Speicher, Inhaltsbibliothek und VM-Klassen. Diese Konfiguration wird von TKG-Dienst-Clustern geerbt, die in diesem Namespace ausgeführt werden.

Lesen Sie als Nächstes die folgenden Themen:

- Verwenden von vSphere-Namespaces mit TKG-Dienstclustern
- Erstellen eines vSphere-Namespace für das Hosting von TKG-Dienst-Clustern
- Konfigurieren eines vSphere-Namespace für TKG-Dienst-Cluster
- Überschreiben der Einstellungen für das Arbeitslastennetzwerk für einen vSphere-Namespace
- Verwenden von VM-Klassen mit TKG-Dienstclustern
- Überprüfen der vSphere-Namespace-Bereitschaft zum Hosten von TKG-Dienstclustern
- Aktivieren der vSphere-Namespace-Erstellung mithilfe von Kubectl
- Entfernen einer vSphere-Namespace

Verwenden von vSphere-Namespaces mit TKG-Dienstclustern

Bei einem vSphere-Namespace handelt es sich um netzwerkbasierte Mandantenfähigkeit auf Supervisor. vSphere-Namespaces werden zum Hosten von TKG-Dienstclustern verwendet und bieten Netzwerkfunktionen, Rollenberechtigungen, dauerhaften Speicher, Ressourcenkontingente sowie eine Integration von Inhaltsbibliothek und VM-Klasse.

vSphere-Namespace Netzwerk

Ein vSphere-Namespace-Netzwerk ist ein Subnetz, das aus dem **Supervisor > Arbeitslastnetzwerk > Namespace-Netzwerk** getrennt wird. Das **Namespace-Subnetzpräfix** definiert die Größe des für jeden vSphere-Namespace reservierten Subnetzes. Die Standardeinstellung ist „/28“.

Das vSphere-Namespace-Netzwerk bietet Konnektivität für TKG-Cluster zu Supervisor. Standardmäßig verwendet der vSphere-Namespace die Netzwerkkonfigurationen der Clusterebene und teilt IP-Adressen aus seinem Subnetz zu. Wenn Sie einen vSphere-Namespace erstellen, werden ein /28-Overlaysegment und der entsprechende IP-Pool für Dienst-Pods in diesem vSphere-Namespace instanziiert.

Wenn der erste TKG-Cluster in einem vSphere-Namespace bereitgestellt wird, verwendet der TKG-Cluster dasselbe Subnetz wie der vSphere-Namespace. Für jeden nachfolgenden TKG-Cluster, der in diesem vSphere-Namespace bereitgestellt wird, wird ein neues Subnetz für diesen Cluster erstellt und mit seinem vSphere-Namespace-Gateway verbunden.

Es gibt eine gemeinsam genutzte Lastausgleichsdienst-Instanz im vSphere-Namespace, die für das Routing des kubectl-Datenverkehrs zu jeder TKG-Clustersteuerungsebene zuständig ist. Darüber hinaus wird für jeden im TKG-Cluster zur Verfügung stehenden Lastausgleichsdienst des Kubernetes-Diensts eine Schicht-4-Lastausgleichsdienst-Instanz für diesen Dienst erstellt.

TKG-Cluster innerhalb desselben vSphere-Namespace nutzen für Nord-Süd-Konnektivität eine SNAT-IP gemeinsam. Für die Ost-West-Konnektivität zwischen Namespaces wird kein SNAT verwendet.

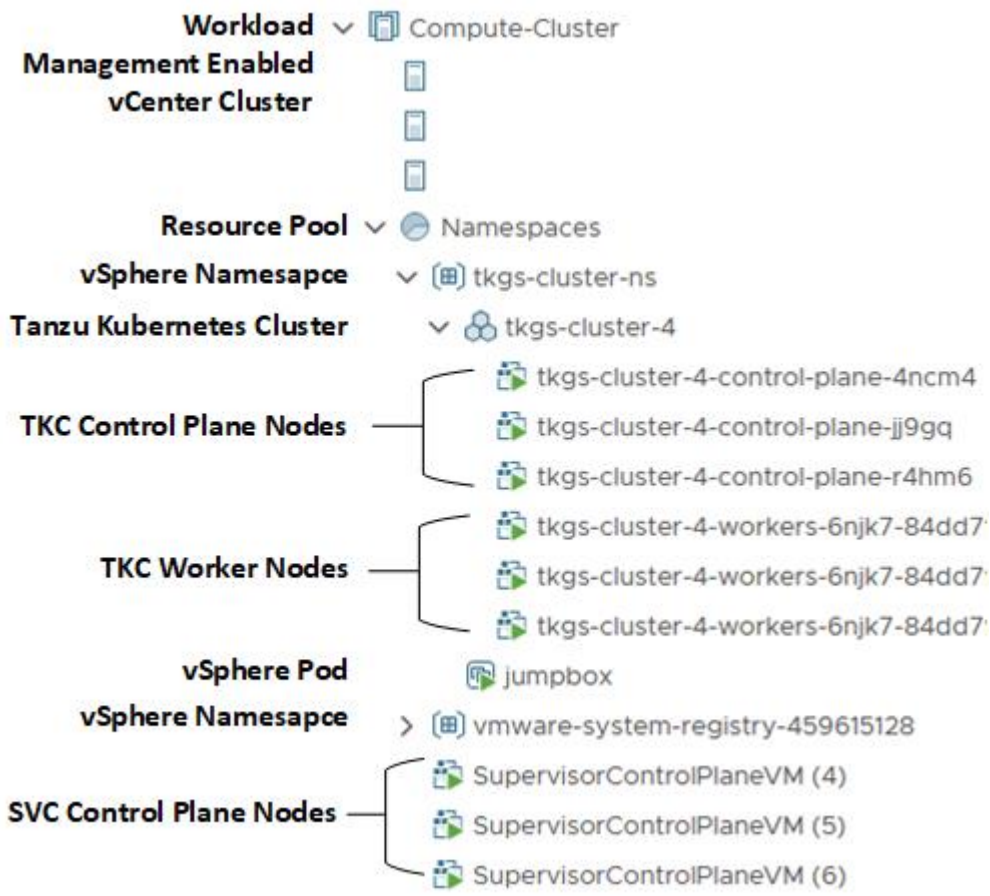
Der vSphere-Namespace ist in der Regel nicht routingfähig. Wenn Sie jedoch ein NSX-Netzwerk verwenden, können Sie das vSphere-Namespace-Netzwerk mit einem routingfähigen Subnetz überschreiben. Weitere Informationen finden Sie unter Überschreiben der Einstellungen für das Arbeitslastennetzwerk für einen vSphere-Namespace.

vSphere-Namespace-Ressourcenpools

Wenn Sie in einer einzelnen Supervisor-Bereitstellung von vSphere-Zonen einen vSphere-Namespace erstellen, wird ein Ressourcenpool erstellt, der diesem Namespace zugrunde liegt. Der vSphere-Namespace dient als logische Einheit von Ressourcen auf Supervisor, einschließlich Computing, Speicher, Berechtigungen, Klassen und Images. Wenn Sie beispielsweise einen CPU- oder Arbeitsspeichergrenzwert auf einem vSphere-Namespace konfigurieren, werden dieselben Ressourcengrenzwerte auf den Ressourcenpool angewendet, der diesem Namespace zugrunde liegt. Auf diese Weise ermöglichen vSphere-Namespaces die Mehrmandantenfähigkeit in Supervisor.

Dieselbe Mehrmandantenerfahrung ist auch für den in drei verschiedenen vSphere-Zonen bereitgestellten Supervisor verfügbar. Wenn ein vSphere-Namespace auf einem Zonen-Supervisor erstellt wird, erstellt das System einen Ressourcenpool in jedem der vSphere-Cluster, die diesen Supervisor unterstützen. Dadurch kann ein in diesem vSphere-Namespace bereitgestellter TKG-Cluster in jeder der zu diesem Supervisor gehörenden Zonen bereitgestellt werden.

Mit vSphere Client können Sie den Ressourcenpool und die Objekte des vSphere-Namespace anzeigen, indem Sie die Perspektive **Hosts und Cluster** und außerdem die Ansicht **VMs und Vorlagen** auswählen. Wenn Sie einen TKG-Cluster bereitstellen, wird er im Ziel-vSphere-Namespace erstellt. Bei einer Supervisor-Zonenbereitstellung wird in jedem vSphere-Cluster derselbe Ressourcenpool verwendet.



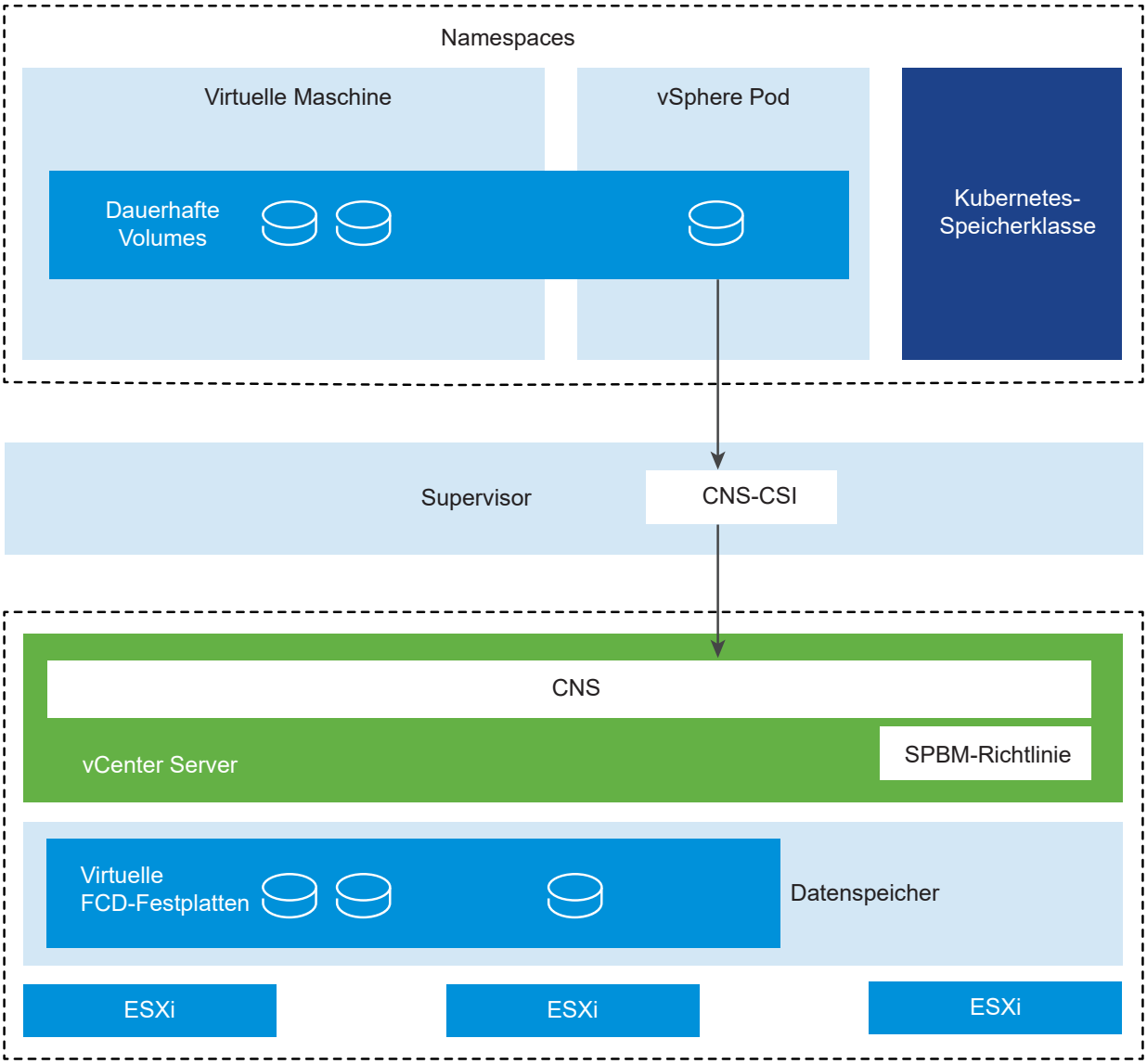
vSphere-Namespace-Speicher für TKG-Dienstcluster

vSphere Cloud Native Storage (CNS) bietet Speicherrichtlinien, die die Bereitstellung dauerhafter Volumes und der zugehörigen virtuellen Festplatten für die Verwendung mit Kubernetes-Arbeitslasten unterstützen.

Die Container Storage Interface (CSI) ist ein Branchenstandard, den Kubernetes zur Bereitstellung von dauerhaftem Speicher für Container verwendet. Der Supervisor führt einen CNS-CSI-Treiber aus, der vSphere CNS-Speicher über den vSphere-Namespace mit der Kubernetes-Umgebung verbindet. vSphere CNS-CSI kommuniziert direkt mit der CNS-Steuerungsebene für alle Speicher-Bereitstellungsanforderungen, die von TKG-Clustern im vSphere-Namespace stammen.

TKG-Cluster führen eine geänderte Version des vSphere CNS-CSI-Treibers aus, der für alle speicherbezogenen Anforderungen verantwortlich ist, die vom TKG-Cluster stammen. Die Anforderungen werden an CNS-CSI auf Supervisor übermittelt und von dort an CNS auf dem vCenter Server weitergeleitet.

Das Diagramm zeigt die Beziehung zwischen vSphere-Namespace, Supervisor und TKG-Clusterspeichermechanismen.



Dauerhafte Speicher-Volumes für TKG-Dienstcluster

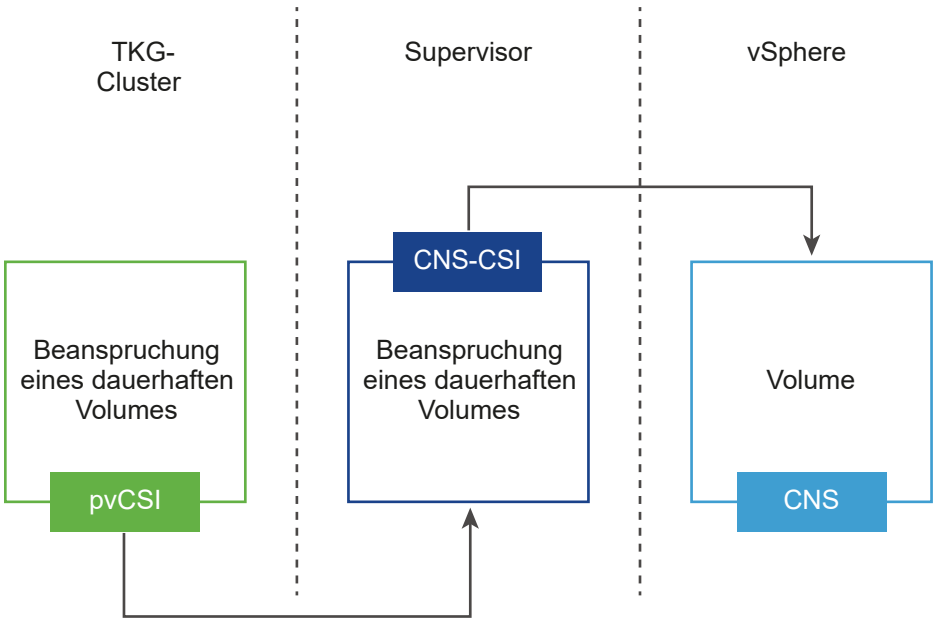
Dauerhafte Volumes sind für statusbehaftete Anwendungen in Kubernetes erforderlich. Weitere Informationen zu dauerhaften Volumes finden Sie in der Kubernetes-Dokumentation.

In der vSphere-Umgebung werden die Objekte eines dauerhaften Volumes von virtuellen Festplatten gesichert, die sich in Datenspeichern befinden. Datenspeicher werden durch Speicherrichtlinien dargestellt. Wenn Sie einem vSphere-Namespace eine vSphere-Speicherrichtlinie zuweisen, wird die Speicherrichtlinie als Kubernetes-Speicherklasse für jeden TKG-Cluster in diesem Namespace zur Verfügung gestellt.

TKG unterstützt die dynamische und statische Bereitstellung persistenter Volumes. Bei dynamischer Bereitstellung muss das dauerhafte Volume nicht vorab bereitgestellt werden. Sie können eine Beanspruchung eines dauerhaften Volumes ausgeben, die auf eine im vSphere-Namespace verfügbare Speicherklasse verweist. TKG stellt automatisch das entsprechende dauerhafte Volume mit einer zugrunde liegenden virtuellen Festplatte bereit. Weitere Informationen finden Sie unter Dynamisches Erstellen dauerhafter Speicher-Volumes.

Bei der statischen Bereitstellung verwenden Sie ein vorhandenes Speicherobjekt und stellen es einem Cluster zur Verfügung. Zur Definition des dauerhaften Volumes stellen Sie die Details des vorhandenen Speicherobjekts, die unterstützten Konfigurationen und Bereitstellungsoptionen bereit. Weitere Informationen finden Sie unter Statisches Erstellen persistenter Speicher-Volumes.

Das Diagramm veranschaulicht den Workflow für die Bereitstellung dynamischer dauerhafter Volumes. Sie erstellen eine PVC mithilfe von `kubectl` im TKG-Cluster. Diese Aktion generiert eine übereinstimmende PVC auf Supervisor und löst den CNS-CSI-Treiber aus, der die CNS-API zum Erstellen von Volumes aufruft.



Speicherklasseneditionen für TKG-Dienstcluster

Um einen vSphere-Namespace zu konfigurieren, weisen Sie eine oder mehrere vSphere-Speicherrichtlinien zu. Bei der Anwendung der vSphere-Speicherrichtlinie wird sie in eine Kubernetes-Speicherklasse konvertiert und auf dem Supervisor repliziert. Darüber hinaus repliziert der TKG-Controller die Speicherklasse auf jedem in diesem vSphere-Namespace bereitgestellten TKG-Cluster.

Auf der TKG-Clusterseite werden zwei Editionen der Speicherklasse angezeigt: eine mit dem benutzerdefinierten Namen, der beim Erstellen der vSphere-Speicherrichtlinie angegeben wurde, und die andere mit dem Suffix `*-latebinding`, das an den Namen angehängt wird.

Die Edition mit später Bindung der Speicherklasse kann von Entwicklern verwendet werden, um eine Bindung an ein dauerhaftes Speicher-Volume durchzuführen, nachdem der Computing-Knoten vom TKG-Pod-Scheduler ausgewählt wurde. Weitere Informationen, wann welche Speicherklasse verwendet werden sollte, finden Sie unter Verwenden von Speicherklassen für persistente Volumes.

```
kubectl get sc
NAME                                    PROVISIONER              RECLAIMPOLICY
VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
wcpglobal-storage-profile               csi.vsphere.vmware.com   Delete
Immediate              true                   2m43s
wcpglobal-storage-profile-latebinding   csi.vsphere.vmware.com   Delete
WaitForFirstConsumer   true                   2m43s
```

vSphere-Namespace-Erstellung

Es gibt mehrere Möglichkeiten, einen vSphere-Namespaces zu erstellen.

Administratoren können vSphere-Namespaces mithilfe des vSphere Client erstellen. Weitere Informationen finden Sie unter Erstellen eines vSphere-Namespace für das Hosting von TKG-Dienst-Clustern.

vCenter Single Sign-On-Benutzer, denen die Rollenberechtigung **Besitzer** für einen vSphere-Namespace erteilt wurde, können vSphere-Namespaces mithilfe von `kubectl` per Self-Service erstellen. Weitere Informationen finden Sie unter Aktivieren der vSphere-Namespace-Erstellung mithilfe von Kubectl.

VMware stellt neben vCenter Server-APIs für die Verwaltung des Lebenszyklus von vSphere-Namespaces auch SDKs (Software Development Kits) bereit, einschließlich:

- Die Namespace-Verwaltungs-APIs stellen REST-basierte Ressourcen für die Verwaltung von vSphere-Namespaces bereit.
- Die Namespaces-APIs stellen REST-basierte Ressourcen für die Verwaltung der Zugriffssteuerung von Themen auf vSphere-Namespaces bereit.
- Das vSphere Automation SDK for Java stellt mehrere Pakete zur Automatisierung der Erstellung und Lebenszyklusverwaltung von vSphere-Namespaces bereit.
- Ebenso bietet das vSphere Automation SDK for Python mehrere Pakete für die Automatisierung der Erstellung und der Lebenszyklusverwaltung von vSphere-Namespaces.

Erstellen eines vSphere-Namespace für das Hosting von TKG-Dienst-Clustern

Sie stellen TKG-Dienst-Cluster in einem vSphere-Namespace bereit.

Erstellen Sie einen vSphere-Namespace, um einen oder mehrere TKG-Dienst Cluster zu hosten.

Verfahren

1 Melden Sie sich mithilfe des vSphere Client bei vCenter Server an.

2 Wählen Sie**Arbeitslastverwaltung > Namespaces > Neuer Namespace** aus.

3 Wählen Sie den vSphere-Cluster aus, bei dem die Option Supervisor aktiviert ist.

4 Geben Sie einen Namen für die vSphere-Namespace-VM ein.

Der Name muss in einem DNS-konformen Format vorliegen und für alle von vCenter Server verwalteten Supervisor-Instanzen eindeutig sein.

5 Geben Sie eine Beschreibung für den vSphere-Namespace ein.

6 Geben Sie das **Namespace-Netzwerk** auf Basis des Typs des Netzwerk-Stacks an, der für Supervisor verwendet wird.

Netzwerkstack	
vSphere-Netzwerk	Wählen Sie im Menü **Netzwerk** ein Arbeitslastnetzwerk.
NSX-Netzwerk	Das Netzwerk wird von Supervisor vererbt. In diesem Fall ist keine Aktion erforderlich. Oder wählen Sie **Cluster-Netzwerkeinstellungen außer Kraft setzen** aus und passen Sie das Netzwerk für diesen vSphere-Namespace an. Weitere Informationen finden Sie unter Überschreiben der Einstellungen für das Arbeitslastennetzwerk für einen vSphere-Namespace.

7 Klicken Sie auf **Erstellen**, um den vSphere-Namespace zu erstellen.

Der Namespace wird im Supervisor erstellt.

8 Konfigurieren Sie den vSphere-Namespace für TKG-Cluster. Weitere Informationen finden Sie unter Konfigurieren eines vSphere-Namespace für TKG-Dienst-Cluster.

Konfigurieren eines vSphere-Namespace für TKG-Dienst-Cluster

Sie stellen einen oder mehrere TKG-Dienst-Cluster in einem vSphere-Namespace bereit. Auf den vSphere-Namespace angewendete Konfigurationseinstellungen werden von jedem hier bereitgestellten TKG-Dienst-Cluster übernommen.

Konfigurieren von Rollenberechtigungen für den vSphere-Namespace

Rollenberechtigungen sind auf den vSphere-Namespace beschränkt. Drei Rollenberechtigungen können Benutzern und Gruppen des TKG-Clusters zugewiesen werden: **Besitzer**, **Schreibzugriff** und **Lesezugriff**. In der Tabelle werden die einzelnen Rollen beschrieben. Weitere Informationen finden Sie unter Informationen zur Identitäts- und Zugriffsverwaltung für TKG-Dienst-Cluster.

Wenn Sie vCenter Single Sign-On verwenden, sind alle drei Rollen verfügbar. Informationen zum Zuweisen von SSO-Benutzern und -Gruppen zu einem vSphere-Namespace finden Sie unter Konfigurieren von vSphere-Namespace-Berechtigungen für Benutzer und Gruppen mit vCenter Single Sign-On.

Bei Verwendung eines externen OIDC-Anbieters steht die Rollenberechtigung **Besitzer** nicht zur Verfügung. Informationen zum Zuweisen von OIDC-Benutzern und -Gruppen zu einem vSphere-Namespace finden Sie unter Konfigurieren von vSphere-Namespace-Berechtigungen für Benutzer und Gruppen externer Identitätsanbieter.

Rolle	Beschreibung
Besitzer	Mit der Rollenberechtigungen und -bindungen können zugewiesene Benutzer und Gruppen vSphere-Namespace-Objekte mithilfe von kubectl verwalten und TKG-Cluster betreiben. Weitere Informationen hierzu finden Sie unter Aktivieren der vSphere-Namespace-Erstellung mithilfe von Kubectl.
Schreibzugriff	Mit der Rollenberechtigungen und -bindungen können zugewiesene Benutzer und Gruppen vSphere-Namespace-Objekte anzeigen und TKG-Cluster betreiben. Ein vCenter Single Sign-On-Benutzer oder eine entsprechende Gruppe, der die Berechtigung **Schreibzugriff** gewährt wurde, ist an die Kubernetes-Rolle **cluster-admin** für jeden in diesem vSphere-Namespace bereitgestellten TKG-Cluster gebunden.
Lesezugriff	Mit der Rollenberechtigungen und -bindungen können zugewiesene Benutzer und Gruppen vSphere-Namespace-Objekte anzeigen. **Hinweis** In Kubernetes ist keine gleichwertige schreibgeschützte Rolle vorhanden, an die die Berechtigung **Lesezugriff** gebunden werden kann. Informationen zum Gewähren von Clusterzugriff für Kubernetes-Benutzer finden Sie unter Gewähren von vCenter SSO-Zugriff auf TKG-Dienst-Cluster für Entwickler.

Konfigurieren eines dauerhaften Speichers für den vSphere-Namespace

Sie können einem vSphere-Namespace eine oder mehrere vSphere-Speicherrichtlinien zuweisen. Eine zugewiesene Speicherrichtlinie steuert die Datenspeicherplatzierung dauerhafter Volumes in der vSphere-Speicherumgebung.

In der Regel definiert ein vSphere-Administrator eine vSphere-Speicherrichtlinie. Wenn Sie vSphere-Zonen verwenden, muss die Speicherrichtlinie mit der `Zonal`-Topologie konfiguriert werden. Weitere Informationen finden Sie unter Erstellen einer vSphere-Speicherrichtlinie für TKG-Dienst-Cluster.

So weisen Sie einem vSphere-Namespace eine vSphere-Speicherrichtlinie zu:

1 Wählen Sie **Arbeitslastverwaltung > Namespace** und den zielseitigen vSphere-Namespace aus.

2 Wählen Sie auf der Kachel **Speicher** die Option **Speicher hinzufügen** aus.

3 Wählen Sie eine oder mehrere Speicherrichtlinien aus den verfügbaren Optionen aus.

Für jede vSphere-Speicherrichtlinie, die Sie einem vSphere-Namespace zuweisen, erstellt das System zwei übereinstimmende Kubernetes-Speicherklassen in diesem vSphere-Namespace. Diese Speicherklassen werden auf jeden in diesem vSphere-Namespace bereitgestellten TKG-Cluster repliziert. Weitere Informationen finden Sie unter Verwenden von Speicherklassen für persistente Volumes.

Festlegen von Kapazitäts- und Nutzungsgrenzwerten für den vSphere-Namespace

Wenn Sie einen vSphere-Namespace konfigurieren, wird auf vCenter Server ein Ressourcenpool für den vSphere-Namespace erstellt. Standardmäßig ist dieser Ressourcenpool ohne Kapazitäts- und Nutzungskontingent konfiguriert. Ressourcen sind durch die Infrastruktur begrenzt.

In der Kachel **Kapazität und Nutzung** für den vSphere-Namespace können Sie die folgenden **Grenzwerte** konfigurieren.

CPU	Die Menge der CPU-Ressourcen, die für den vSphere Namespace reserviert werden soll.
Arbeitsspeicher	Die Menge an Arbeitsspeicher, die für den vSphere Namespace reserviert werden soll.
Speicher	Die Gesamtmenge an Speicherplatz, die für den vSphere Namespace reserviert werden soll.
Grenzwerte für Speicherrichtlinien	Legen Sie die Speichermenge fest, die für jede mit dem vSphere Namespace verknüpfte Speicherrichtlinie einzeln reserviert ist.

In der Regel müssen Sie für TKG-Clusterbereitstellungen kein Ressourcenkontingent für den vSphere-Namespace konfigurieren. Wenn Sie Kontingentgrenzen zuweisen, ist es wichtig, die potenziellen Auswirkungen auf die dort bereitgestellten TKG-Cluster zu verstehen.

CPU- und Arbeitsspeichergrenzwerte

Die für den vSphere-Namespace konfigurierten CPU- und Arbeitsspeichergrenzwerte haben keinen Einfluss auf einen dort bereitgestellten TKG-Cluster, wenn die TKG-Clusterknoten den garantierten VM-Klassentyp verwenden. Wenn die TKG-Clusterknoten jedoch den Verwenden von VM-Klassen mit TKG-Dienstclustern verwenden, können sich die CPU- und Arbeitsspeichergrenzwerte auf den TKG-Cluster auswirken.

Da der bestmögliche VM-Klassentyp eine Überbelegung von Ressourcen zulässt, ist es möglich, dass keine Ressourcen mehr verfügbar sind, wenn Sie CPU- und Arbeitsspeichergrenzwerte für den vSphere-Namespace festgelegt haben, in dem Sie TKG-Cluster bereitstellen. Wenn ein Konflikt auftritt und die Steuerungsebene der TKG-Cluster betroffen ist, wird die Ausführung des Clusters möglicherweise beendet. Deshalb sollten Sie für Produktionscluster immer den Verwenden von VM-Klassen mit TKG-Dienstclustern verwenden. Wenn Sie den garantierten VM-Klassentyp nicht für alle Produktionsknoten verwenden können, sollten Sie den garantierten Klassentyp zumindest für die Knoten der Steuerungsebene verwenden.

Grenzwerte für Speicher- und Speicherrichtlinien

Ein für den vSphere-Namespace konfigurierter Speichergrenzwert bestimmt die Gesamtmenge des Speichers, der dem vSphere-Namespace für alle dort bereitgestellten TKG-Cluster zur Verfügung steht.

Ein für den vSphere-Namespace konfigurierter Grenzwert für Speicherrichtlinien bestimmt die Menge des für diese Speicherklasse verfügbaren Speichers für jeden TKG-Cluster, in dem die Speicherklasse repliziert wird.

Einige Arbeitslasten haben Mindestspeicheranforderungen. Informationen dazu finden Sie beispielsweise unter #unique_112.

Verknüpfen der TKR-Inhaltsbibliothek mit dem TKG-Dienst

Zum Bereitstellen von TKG-Clustern verknüpfen Sie den TKG-Dienst mit einer Inhaltsbibliothek. Informationen zum Erstellen einer Inhaltsbibliothek für das Hosting von TKr-Images finden Sie unter Kapitel 5 Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster.

So verknüpfen Sie eine TKr-Inhaltsbibliothek mit einem vSphere-Namespace:

1 Wählen Sie **Arbeitslastverwaltung > Supervisoren > Supervisor** und dann die Supervisor-Instanz aus.

2 Wählen Sie **Konfigurieren > Supervisor > Allgemein > Tanzu Kubernetes Grid Service** aus.

3 Wählen Sie **Inhaltsbibliothek > Bearbeiten** aus.

4 Wählen Sie eine TKR-Inhaltsbibliothek aus.

5 Navigieren Sie zum vSphere-Namespace und wählen Sie **Namespace verwalten** aus.

6 Stellen Sie sicher, dass die ausgewählte Inhaltsbibliothek im Konfigurationsbereich unter **Tanzu Kubernetes Grid Service** angezeigt wird.

Es ist wichtig zu verstehen, dass die TKR-Inhaltsbibliothek nicht auf den Namespace-Geltungsbereich beschränkt ist. Alle vSphere-Namespaces verwenden dieselbe TKR-Inhaltsbibliothek, die für den Tanzu Kubernetes Grid Service (TKGS) konfiguriert ist. Änderungen an der TKR-Inhaltsbibliothek für TKGS wirken sich auf jeden vSphere-Namespace aus.

Hinweis Die Inhaltsbibliothek, auf die in der Kachel **VM-Dienst** verwiesen wird, ist für die Verwendung mit eigenständigen VMs und nicht für die TKR-Inhaltsbibliothek vorgesehen. Fügen Sie die TKR-Inhaltsbibliothek nicht in dieser Kachel hinzu.

Verknüpfen von VM-Klassen mit dem vSphere-Namespace

vSphere IaaS control plane bietet mehrere standardmäßige Verwenden von VM-Klassen mit TKG-Dienstclustern, und Sie können Ihre eigenen erstellen.

Zum Bereitstellen eines TKG-Clusters verknüpfen Sie eine oder mehrere Verwenden von VM-Klassen mit TKG-Dienstclustern mit dem zielseitigen vSphere-Namespace. Gebundene Klassen stehen für die Verwendung durch TKG-Clusterknoten zur Verfügung, die in diesem vSphere-Namespace bereitgestellt werden.

Um die Standard-VM-Klassen mit einem vSphere-Namespace zu verknüpfen, melden Sie sich mithilfe des vCenter Server -vSphere Client an und führen Sie das folgende Verfahren durch.

1 Wählen Sie **Arbeitslastverwaltung > Namespace** und den zielseitigen vSphere-Namespace aus.

2 Wählen Sie für die Kachel **VM-Dienst** die Option **VM-Klasse hinzufügen** aus.

3 Wählen Sie jede VM-Klasse aus, die Sie hinzufügen möchten.

   a Zum Hinzufügen der Standard-VM-Klassen aktivieren Sie das Kontrollkästchen in der Tabellenkopfzeile auf Seite 1 der Liste, navigieren Sie zu Seite 2 und aktivieren Sie das Kontrollkästchen in der Tabellenkopfzeile auf dieser Seite. Stellen Sie sicher, dass alle Klassen ausgewählt sind.

   b Um eine benutzerdefinierte Klasse zu erstellen, klicken Sie auf **Neue VM-Klasse erstellen**. Anweisungen finden Sie in der Dokumentation zu VM-Diensten.

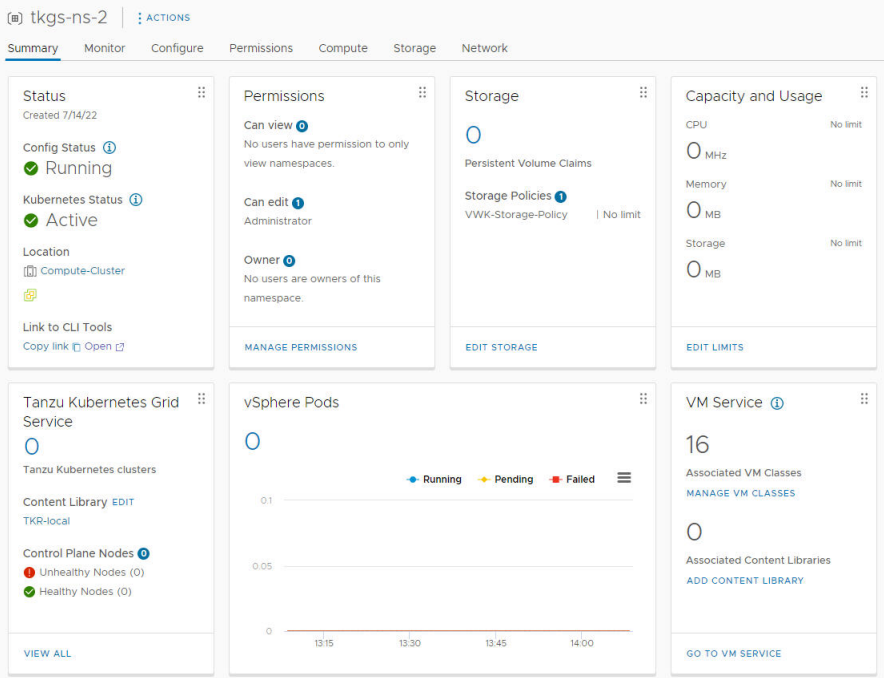
4 Klicken Sie auf **OK**, um den Vorgang abzuschließen.

5 Bestätigen Sie, dass die Klassen hinzugefügt wurden. Die Kachel **VM-Dienst** zeigt **VM-Klassen verwalten** an.

Überprüfen der vSphere-Namespace-Konfiguration

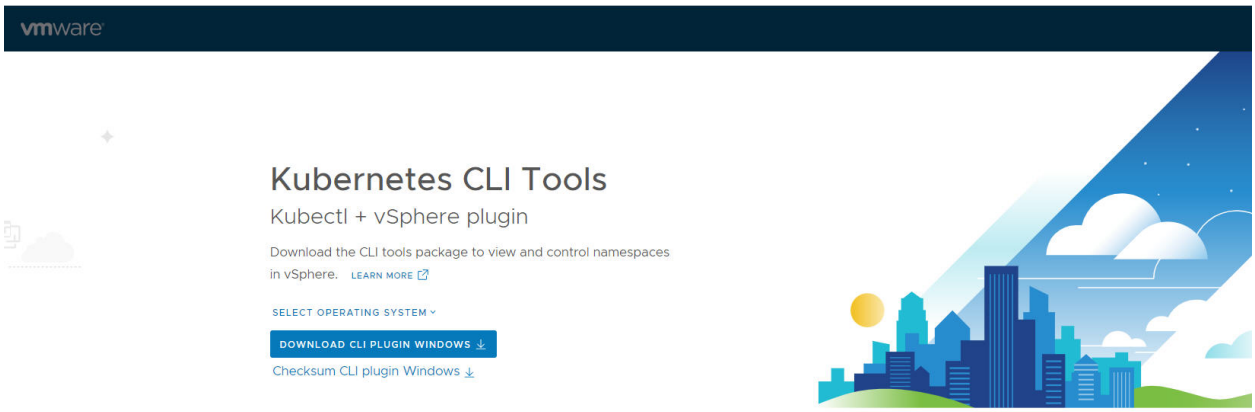
Ein konfigurierter vSphere-Namespace umfasst **Status**, **Berechtigungen**, **Speicher**, **Kapazität und Nutzung**, **TKR-Inhaltsbibliothek** und **VM-Klassen**.

Abbildung 6-1. Konfiguriert vSphere-Namespace



Die Kachel **Status** enthält einen Link zu den vSphere IaaS control plane CLI-Tools. Die Seite „DevOps“ wird vom Load Balancer der Supervisor-Steuerungsebene verwaltet. Stellen Sie TKG-Clusterbenutzern den Link zum Herunterladen der Kubernetes-CLI-Tools für vSphere zur Verfügung. Weitere Informationen finden Sie unter Installieren des Kubernetes-CLI-Tools für vSphere.

Abbildung 6-2. Seite „DevOps“ für vSphere-Namespace



Informationen dazu, wie Sie die vSphere-Namespace-Konfiguration mithilfe von kubectl überprüfen, finden Sie unter Überprüfen der vSphere-Namespace-Bereitschaft zum Hosten von TKG-Dienstclustern.

Überschreiben der Einstellungen für das Arbeitslastennetzwerk für einen vSphere-Namespace

Wenn Sie Supervisor bereitstellen, konfigurieren Sie die Standardeinstellungen für das Arbeitslastnetzwerk. Wenn Sie den Supervisor mit einem NSX-Netzwerk aktiviert haben, können Sie beim Erstellen eines vSphere-Namespaces die standardmäßigen Einstellungen für das Arbeitslastennetzwerk außer Kraft setzen. Überschriebene Einstellungen für das Arbeitslastennetzwerk gelten nur für dieses vSphere-Namespace-Segment.

Überschreiben der Einstellungen für das Arbeitslastennetzwerk (nur NSX)

Wenn Sie einen vSphere-Namespace erstellen, wird ein Netzwerksegment erstellt. Standardmäßig wird dieses Netzwerksegment von dem Arbeitslastnetzwerk abgeleitet, das im Supervisor konfiguriert ist. Weitere Informationen hierzu finden Sie unter vSphere-Namespace Netzwerk.

Wenn der Supervisor mit einem NSX-Netzwerk konfiguriert ist, können Sie während der Erstellung eines vSphere-Namespace die Option **Cluster-Netzwerkeinstellungen außer Kraft setzen** für den vSphere-Namespace auswählen. Wenn Sie diese Option auswählen, können Sie das vSphere-Namespace-Netzwerk anpassen, indem Sie den Feldern Ingress, Egress und Namespace-Netzwerk CIDRs hinzufügen. Die neuen CIDRs, die Sie hinzufügen, setzen vorhandene CIDRs für diese vSphere-Namespace-Instanz außer Kraft.

Wenn Sie NSX Version 4.1.1 oder höher konfiguriert haben und NSX Advanced Load Balancer Version 22.1.4 oder höher mit einer Enterprise-Lizenz für NSX installiert, konfiguriert und registriert ist, wird NSX Advanced Load Balancer als Lastausgleichsdienst mit NSX verwendet. Wenn Sie Versionen von NSX vor 4.1.1 konfiguriert haben, wird der NSX-Lastausgleichsdienst verwendet. Weitere Informationen finden Sie unter Supervisor-Netzwerk im Thema *Konzepte und Planung der vSphere IaaS-Steuerungsebene*.

Der typische Anwendungsfall für die Außerkraftsetzung von Supervisor-Netzwerkeinstellungen ist die Bereitstellung eines TKG-Clusters mit routingfähigem Pod-Netzwerk. Weitere Informationen zur richtigen Vorgehensweise sowie Links zu Beispielen finden Sie in den Konfigurationseinstellungen.

Tabelle 6-1. Überlegungen zur vSphere-Namespace-Netzwerkplanung

Überlegungen	Beschreibung
NSX erforderlich	Um Supervisor-Netzwerkeinstellungen für einen bestimmten vSphere-Namespace außer Kraft zu setzen, muss der Supervisor mit einem NSX-Netzwerk konfiguriert sein.
NSX-Installation	Um Supervisor-Netzwerkeinstellungen für einen bestimmten vSphere-Namespace außer Kraft zu setzen, muss die Installation von NSX einen für Tier-0-Gateways (Router) dedizierten Edge-Cluster und einen anderen für Tier-1-Gateways dedizierten Edge-Cluster enthalten. Weitere Informationen finden Sie im Installationshandbuch für NSX *Installieren und Konfigurieren von vSphere with Tanzu*.
IPAM erforderlich	Wenn Sie Supervisor-Netzwerkeinstellungen für einen bestimmten vSphere-Namespace außer Kraft setzen, muss das neue vSphere-Namespace-Netzwerk Subnetze für Ingress, Egress und Namespace-Netzwerk angeben, die auf dem Supervisor und im Vergleich zu anderen vSphere-Namespace-Netzwerken eindeutig sind. Sie müssen die Zuteilung von IP-Adressen entsprechend verwalten.
Supervisor-Routing	Der Supervisor muss direkt zu den TKG-Clusterknoten und Ingress-Subnetzen weitergeleitet werden können. Bei Auswahl eines Tier-0-Gateways für den vSphere-Namespace haben Sie zwei Optionen zum Konfigurieren des erforderlichen Routings: ■ Verwenden Sie ein VRF-Gateway (Virtual Routing and Forwarding), um die Konfiguration vom Tier-0-Gateway vom Supervisor zu übernehmen ■ Verwenden Sie das Border Gateway Protocol (BGP) zum Konfigurieren von Routen zwischen dem Tier-0-Gateway vom Supervisor und dem dedizierten Tier-0-Gateway Weitere Informationen zu diesen Optionen finden Sie in der Dokumentation zu NSX Tier-0-Gateways.

Konfigurationsfelder, um Supervisor-Netzwerkeinstellungen außer Kraft zu setzen.

Tabelle 6-2. vSphere-Namespace – Konfigurationsoptionen für das Überschreiben der Einstellungen für das Arbeitslastennetzwerk

Komponente	Konfiguration
Tier-0-Gateway	Das NSX-Tier-0-Gateway verbindet Supervisor mit dem physischen Netzwerk. Die ausgewählte Tier-0-Gateway wird mit dem Tier-1-Gateway verknüpft, das für den vSphere-Namespace erstellt wird. Durch die Auswahl eines neuen Tier-0-Gateways wird das bei der Aktivierung von Supervisor konfigurierte Tier-0-Gateway außer Kraft gesetzt. In diesem Fall müssen Sie neue CIDR-Bereiche konfigurieren. Wenn Sie ein VRF-Gateway auswählen, das mit dem Tier-0-Gateway verknüpft ist, werden das Netzwerk und die Subnetze automatisch konfiguriert. Nachdem Sie ein Tier-0-Gateway ausgewählt und die Konfiguration abgeschlossen haben, können Sie das Tier-0-Gateway nicht mehr ändern.
Größe des Lastausgleichsdiensts	Wählen Sie die Größe der Load Balancer-Instanz auf dem Tier-1-Gateway für den vSphere-Namespace aus. Legen Sie die Größe des Load Balancers auf klein (Standard), mittel oder groß fest. Es kann nur eine festgelegte Anzahl von Load Balancer-Instanzen pro Edge-Knoten definiert werden. Weitere Informationen finden Sie unter Maximalwerte für die Konfiguration. **Hinweis** Diese Einstellung gilt nicht für NSX Advanced Load Balancer.
NAT-Modus	Der NAT-Modus ist standardmäßig aktiviert. Dies bedeutet, dass das Subnetz des Namespace-Netzwerks voraussichtlich nicht routingfähig ist und Sie Namespace-Netzwerk-, Ingress- und Egress-CIDRs konfigurieren müssen. Wenn Sie die Auswahl des NAT-Modus aufheben, wird das System darüber informiert, dass Sie einen routingfähigen CIDR-Bereich für das Namespace-Netzwerk bereitstellen werden. Wenn Sie den NAT-Modus deaktivieren, sind die IP-Adressen des TKG-Clusterknotens direkt von außerhalb des Tier-0-Gateways zugänglich, und Sie müssen Egress-CIDR nicht konfigurieren. Um einen Cluster ohne NAT-Modus bereitzustellen, deaktivieren Sie den NAT-Modus und sehen Sie sich die Beispiele an: Kapitel 7 Bereitstellen von TKG-Dienstclustern.
Namespace-Netzwerk-CIDR	Beim Namespace-Netzwerk-CIDR handelt es sich um ein Subnetz, das als IP-Pool betrieben wird, wobei das Subnetzpräfix des Namespace die Größe eines beliebigen nachfolgenden CIDR-Blocks beschreibt, der aus diesem IP-Pool entfernt wird. Jedes Mal, wenn ein vSphere-Namespace erstellt wird, wird ein Subnetz des Namespace-Netzwerks zugeteilt. Die subnetzspezifische Größe dieses Blocks beträgt /24, was bedeutet, dass maximal 256 Pods pro vSphere-Namespace erstellt werden können. Weitere Informationen finden Sie unter „Maximalwerte für die Konfiguration“. Das Namespace-Netzwerk-CIDR wird verwendet, um TKG-Clustern, die an vSphere-Namespace-Segmente angefügt sind, IP-Adressen zuzuteilen. Wenn der NAT-Modus ausgewählt ist, ist das CIDR voraussichtlich nicht routingfähig. Wenn der NAT-Modus deaktiviert ist, muss das Namespace-Netzwerk-CIDR routingfähig sein.

Tabelle 6-2. vSphere-Namespace – Konfigurationsoptionen für das Überschreiben der Einstellungen für das Arbeitslastennetzwerk (Fortsetzung)

Komponente	Konfiguration
Namespace-Subnetzpräfix	Geben Sie das Subnetzpräfix ein, das die Größe des für Namespace-Segmente reservierten Subnetzes angibt. Der Standardwert ist „28“. Das Präfix des Namespace-Subnetzes definiert das IP-Subnetz, das für jedes vSphere-Namespace-Segment erstellt wurde. Beispielsweise würde das Festlegen eines /24-Präfixes zu einem vSphere-Namespace-Segment mit einem IP-Subnetz mit 254 IP-Adressen führen, das den dort bereitgestellten TKG-Clustern zugeteilt werden soll. Zusätzliches Beispiel: Namespace-Netzwerk-CIDR = 192.168.1.0/24 Namespace-Subnetzpräfix = /28 In diesem Fall kann TKGS 16x 192.168.1.x/28-CIDR-Blöcke aus dem Subnetz 192.168.1.0/24 bereitstellen. Dies ermöglicht die Instanziierung von 16 TKGS-Namespace-Netzwerken, mit denen Ihre TKG-Dienst-verwalteten VMs (TKC, VMS, vSphere-Pods) verbunden sind. Beispielsweise erhält jede TKC ein dediziertes Namespace-CIDR. Dieses könnte in diesem Fall auf 192.168.1.0/28, während das nächste TKC-Namespace-Subnetz auf 192.168.0.16/28 usw. lauten könnte.
Ingress	Der Ingress-IP-CIDR-Block wird verwendet, um IP-Adressen für Kubernetes-Dienste zuzuteilen, die von einem Load Balancer des Diensttyps und von einem Ingress-Controller in allen vSphere-Namespaces veröffentlicht werden. TKG-Clusterdienste und Ingress erhalten ihre IP-Adressen von diesem CIDR-Block. Geben Sie eine CIDR-Anmerkung ein, die den Ingress-IP-Bereich für die virtuellen IP-Adressen festlegt, die vom Lastausgleichsdienst für TKG-Cluster veröffentlicht werden. **Hinweis** Diese Einstellung gilt nicht für NSX Advanced Load Balancer.
Egress	Egress-IP-CIDR wird verwendet, um IP-Adressen für SNAT (Source Network Address Translation) für den Datenverkehr zuzuweisen, der den vSphere-Namespace verlässt, um auf externe Dienste zuzugreifen. Geben Sie eine CIDR-Anmerkung ein, die den Egress-IP-Bereich für die SNAT-IP-Adressen bestimmt.

Verwenden von VM-Klassen mit TKG-Dienstclustern

Für die Größe der TKG-Dienstclusterknoten geben Sie die VM-Klasse (virtuelle Maschine) an. Die Plattform bietet VM-Standardklassen, aber Sie können auch eigene Klassen erstellen. Zur Verwendung einer VM-Klasse ordnen Sie sie dem Ziel-vSphere-Namespace zu und erstellen einen Verweis auf die Klasse im Clustermanifest.

Informationen zu VM-Klassen

Eine VM-Klasse ist eine Anforderung für Ressourcenreservierungen für die Verarbeitungsleistung auf der virtuellen Maschine (VM), einschließlich CPU und Arbeitsspeicher (RAM). Beispielsweise reserviert der VM-Klassentyp namens „guaranteed-large“ 4 CPUs und 16 GB RAM.

Hinweis Die Größe der VM-Festplatte wird durch die OVA-Vorlage und nicht durch die Definition der VM-Klasse festgelegt. Für Tanzu Kubernetes-Versionen beträgt die Festplattengröße 16 GB.

Es gibt zwei Reservierungstypen für VM-Klassen: garantiert und bestmöglich. Die garantierte Klasse reserviert ihre konfigurierten Ressourcen vollständig. Dies bedeutet, dass für einen bestimmten Cluster die Spezifikation `spec.policies.resources.requests` mit der Spezifikation `spec.hardware` übereinstimmt. Die bestmögliche Klasse lässt eine Überbelegung von Ressourcen zu. Für Produktionsarbeitslasten wird empfohlen, den garantierten VM-Klassentyp zu verwenden.

Warnung Da der bestmögliche VM-Klassentyp eine Überbelegung von Ressourcen zulässt, ist es möglich, dass keine Ressourcen mehr verfügbar sind, falls Grenzwerte für den vSphere-Namespace festgelegt wurden, in dem Sie den TKG-Cluster bereitstellen. Wenn ein Konflikt auftritt und die Steuerungsebene betroffen ist, wird die Ausführung des Clusters möglicherweise beendet. Verwenden Sie aus diesem Grund den garantierten VM-Klassentyp für Produktionscluster. Wenn Sie den garantierten VM-Klassentyp nicht für alle Produktionsknoten verwenden können, verwenden Sie den garantierten Klassentyp mindestens für die Knoten der Steuerungsebene.

Verwenden von VM-Klassen mit TKG-Dienstclustern

Um eine VM-Klasse mit einem TKG-Dienstcluster zu verwenden, muss die VM-Klasse an den vSphere-Namespace gebunden sein, in dem der Cluster bereitgestellt wird. Dazu ordnen Sie die Klasse dem Ziel-Namespace zu. Weitere Informationen finden Sie unter Konfigurieren eines vSphere-Namespace für TKG-Dienst-Cluster.

Um die im Ziel-vSphere-Namespace verfügbaren VM-Klassen aufzulisten, verwenden Sie den Befehl `kubectl get virtualmachineclass`.

Hinweis Bei Problemen mit diesem Befehl finden Sie weitere Informationen unter Beheben von Fehlern bei VM-Klassen.

Definitionen für VM-Klassen sind nicht unveränderlich. Jede VM-Klasse kann bearbeitet werden, einschließlich der Definitionen für Standard-VM-Klassen. Wenn eine VM-Klasse bearbeitet wird, bleiben vorhandene TKG-Clusterknoten davon unberührt. Neue TKG-Cluster verwenden die geänderte Klasse.

Vorsicht Wenn Sie eine VM-Klasse bearbeiten, die von einem TKG-Cluster verwendet wird, und dann diesen Cluster horizontal skalieren, verwenden die neuen Knoten die bearbeitete Klassendefinition, die vorhandenen Knoten jedoch die ursprüngliche Klassendefinition, was zu einer Diskrepanz zwischen Klassen führt.

VM-Standardklassen

In der Tabelle werden die Standardtypen von VM-Klassen aufgelistet, die als Bereitstellungsgrößen für Tanzu Kubernetes-Clusterknoten verwendet werden.

Damit nicht übermäßig Ressourcen gebunden werden, sollten Produktionsarbeitslasten den garantierten Klassentyp verwenden. Um zu vermeiden, dass Ihnen der Arbeitsspeicher ausgeht, sollten Sie die kleine oder extrakleine Klassengröße nicht für Worker-Knoten verwenden, auf denen Sie Arbeitslasten in einer beliebigen Umgebung (Entwicklung, Test oder Produktion) bereitstellen.

Tabelle 6-3. VM-Standardklassen

Klasse	CPU	Arbeitsspeicher (GB)	Reservierte CPU und reservierter Arbeitsspeicher
guaranteed-8xlarge	32	128	Ja
best-effort-8xlarge	32	128	Nein
guaranteed-4xlarge	16	128	Ja
best-effort-4xlarge	16	128	Nein
guaranteed-2xlarge	8	64	Ja
best-effort-2xlarge	8	64	Nein
guaranteed-xlarge	4	32	Ja
best-effort-xlarge	4	32	Nein
guaranteed-large	4	16	Ja
best-effort-large	4	16	Nein
guaranteed-medium	2	8	Ja
best-effort-medium	2	8	Nein
guaranteed-small	2	4	Ja
best-effort-small	2	4	Nein
guaranteed-xsmall	2	2	Ja
best-effort-xsmall	2	2	Nein

Benutzerdefinierte VM-Klassen

vSphere IaaS control plane unterstützt benutzerdefinierte VM-Klassen für die Verwendung mit TKG-Clustern. Nachdem Sie eine benutzerdefinierte VM-Klasse definiert haben, müssen Sie sie dem Ziel-vSphere-Namespace verbinden, bevor Sie sie mit einem Cluster verwenden können. Ausführliche Informationen finden Sie in der Dokumentation zu VM-Diensten.

Überprüfen der vSphere-Namespace-Bereitschaft zum Hosten von TKG-Dienstclustern

Nachdem Sie einen vSphere-Namespace konfiguriert haben, melden Sie sich beim Supervisor an, und überprüfen Sie, ob der vSphere-Namespace zum Hosten von TKG-Dienstclustern bereit ist.

Überprüfen der vSphere-Namespace-Konfiguration für die Bereitstellung eines TKG-Dienstclusters

Führen Sie diese Aufgabe zur Vorbereitung für die Bereitstellung von TKG-Dienstclustern aus, um sicherzustellen, dass der vSphere-Namespace ordnungsgemäß konfiguriert ist.

1 Melden Sie sich beim Supervisor an.

```
kubectl vsphere login --server IP-ADDRESS-SUPERVISOR-CLUSTER --vsphere-username VCENTER-SSO-USERNAME
```

2 Wechseln Sie den Kontext zum zielseitigen vSphere-Namespace, in dem Sie einen oder mehrere TKG-Cluster bereitstellen möchten.

```
kubectl config use-context VSPHERE-NAMESPACE-NAME
```

3 Beschreiben Sie den vSphere-Namespace.

```
kubectl describe ns VSPHERE-NAMESPACE-NAME
```

Dieser Befehl gibt den Namen jeder im vSphere-Namespace verfügbaren Speicherklasse sowie im Ressourcenkontingent zurück.

4 Erstellen Sie eine Liste und eine Beschreibung der verfügbaren Tanzu Kubernetes-Versionen.

```
kubectl get tanzukubernetesreleases
```

Dieser Befehl gibt die TKRs zurück, die sich in der für den vSphere-Namespace konfigurierten Inhaltsbibliothek befinden und mit der Bibliothek synchronisiert oder in die Bibliothek hochgeladen wurden.

5 Erstellen Sie eine Liste der verfügbaren VM-Klassen.

```
kubectl get virtualmachineclass
```

Dieser Befehl gibt die VM-Klassen zurück, die mit dem Namespace verknüpft sind. Nur gebundene VM-Klassen können zur Bereitstellung von TKG-Dienstclusterknoten verwendet werden.

Aktivieren der vSphere-Namespace-Erstellung mithilfe von Kubectl

Sie können den vSphere-Namespace-Dienst aktivieren, damit Entwickler den Lebenszyklus von vSphere-Namespaces mithilfe von `kubectl` verwalten können.

Durch die Aktivierung des vSphere-Namespace-Diensts auf Supervisor können Entwickler, die der Besitzerrolle in einem vSphere-Namespace zugewiesen sind, ihre eigenen vSphere-Namespaces mithilfe des Befehls `kubectl create namespace <NAME>` erstellen.

Wenn Sie den vSphere-Namespace-Dienst aktivieren, definieren Sie eine Namespace-Vorlage und aktivieren sie. Entwickler, die der Besitzerrolle zugewiesen sind, verwenden die Vorlage zum Erstellen von Namespaces.

Verfahren

1 Melden Sie sich mithilfe von vSphere Client bei vCenter Server an.

2 Wählen Sie **Arbeitslastverwaltung > Supervisoren** und dann die Supervisor-Instanz aus.

3 Klicken Sie in der Registerkarte **Konfigurieren** auf **Supervisor > Allgemein**.

4 Wählen Sie **Namespace-Dienst** aus.

5 Klicken Sie auf den Schalter **Status**, um die Funktion zu aktivieren.

Die Seite **Namespace-Vorlage erstellen** wird angezeigt.

6 Konfigurieren Sie im Bereich **Konfiguration** die Ressourceneinschränkungen für den Namespace.

Option	Beschreibung
CPU	Die Menge der CPU-Ressourcen, die für den Namespace reserviert werden soll.
Arbeitsspeicher	Die Menge an Arbeitsspeicher, die für den Namespace reserviert werden soll.
Speicher	Die Gesamtmenge an Speicherplatz, die für den Namespace reserviert werden soll.
Speicherrichtlinie	Legen Sie die Speichermenge fest, die für jede mit dem Namespace verknüpfte Speicherrichtlinie einzeln reserviert ist.
VM-Klassen	Wählen Sie die VM-Klassen aus. Verwenden Sie die Strg-Taste, um mehrere auszuwählen.
Inhaltsbibliothek	Wählen Sie die TKR-Inhaltsbibliothek aus.

7 Klicken Sie auf **Weiter**.

8 Fügen Sie im Bereich **Berechtigungen** DevOps-Ingenieure und -Gruppen hinzu, damit sie die Vorlage zum Erstellen von Namespaces verwenden können.

a Wählen Sie die vsphere.local-Identitätsquelle aus (muss vSphere SSO verwenden).

b Wählen Sie einen Benutzer oder eine Gruppe aus.

c Wählen Sie die Rolle „Besitzer“ aus.

9 Im Bereich **Überprüfen und bestätigen** werden die von Ihnen konfigurierten Eigenschaften angezeigt.

Überprüfen Sie die Eigenschaften und klicken Sie auf **Fertig**.

10 Stellen Sie sicher, dass der vSphere-Namespace-Dienst auf „Aktiv“ festgelegt ist.

Eine vSphere-Namespace-Vorlage ist konfiguriert und befindet sich im Zustand „Aktiv“. vSphere-Namespace-Benutzer/Gruppen, die der Rolle „Besitzer“ zugewiesen sind, können die Vorlage verwenden, um mithilfe des kubectl-Befehls `kubectl create namespace <NAME>` vSphere-Namespaces zu erstellen.

Entfernen einer vSphere-Namespace

Sie können einen vSphere-Namespace vom Supervisor entfernen. Zuvor sollten Sie alle dort bereitgestellten TKG-Dienst-Cluster löschen.

Voraussetzung: Löschen von TKG-Dienst-Clustern im vSphere-Namespace

Bevor Sie einen vSphere-Namespace entfernen, sollten Sie alle dort bereitgestellten TKG 2.0-Cluster löschen.

Hinweis Sie sollten auch alle in der vSphere-Namespace bereitgestellten vSphere-Pods entfernen.

Verwenden Sie kubectl oder die Tanzu-CLI, um einen TKG-Cluster zu löschen. Weitere Informationen finden Sie unter Löschen eines TKG-Clusters mithilfe von Kubectl oder der Tanzu-CLI.

Hinweis Versuchen Sie nicht, einen TKG-Cluster über die vSphere Client- oder vCenter Server-CLI zu löschen.

Entfernen des vSphere-Namespaces

Führen Sie diese Aufgabe zur Vorbereitung für die Bereitstellung von TKG 2.0-Clustern aus, um sicherzustellen, dass der vSphere-Namespace ordnungsgemäß konfiguriert ist.

1 Melden Sie sich über vSphere Client bei vCenter Server an.

2 Wählen Sie Arbeitslastverwaltung aus.

3 Wählen Sie die Registerkarte **Namespace** aus.

Jede vSphere-Namespace, die auf Supervisor erstellt wurde, wird aufgelistet.

4 Wählen Sie den vSphere-Namespace aus, den Sie entfernen möchten.

5 Wählen Sie **Entfernen**.

Das System entfernt den vSphere-Namespace. Es kann etwas Zeit in Anspruch nehmen, bis der Vorgang abgeschlossen ist. Verwenden Sie den Aufgabenbereich, um den Fortschritt zu verfolgen.

Bereitstellen von TKG-Dienstclustern

7

Bei Verwendung des TKG-Diensts können Sie zwei Typen von Arbeitslastclustern erstellen: Tanzu Kubernetes-Cluster und Cluster basierend auf einer ClusterClass.

Lesen Sie als Nächstes die folgenden Themen:

- Über die TKG-Clusterbereitstellung
- Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl
- Workflow zum Bereitstellen von TKG-Clustern auf mithilfe der Tanzu-CLI
- Testen der TKG-Clusterbereitstellung mithilfe von kubectl
- Löschen eines TKG-Clusters mithilfe von Kubectl oder der Tanzu-CLI
- Verwenden der v1beta1-API von Clustern
- Verwenden der v1alpha3-API von TanzuKubernetesCluster

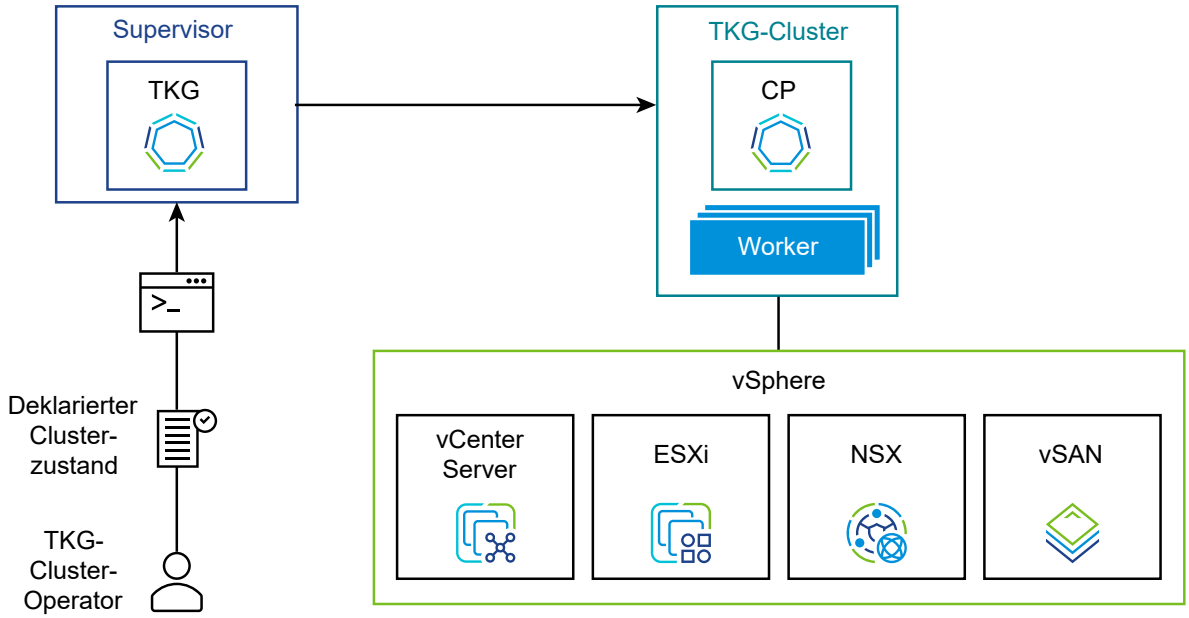
Über die TKG-Clusterbereitstellung

Der TKG-Dienst bietet zwei APIs und unterstützende Clients für die Bereitstellung von TKG-Clustern und deren Lebenszyklus.

Bereitstellung von TKG-Clustern

Das Diagramm zeigt den Workflow für die Bereitstellung von TKG-Clustern auf Supervisor.

Abbildung 7-1. Bereitstellung von TKG-Clustern auf Supervisor



TKG-Clustertypen

Es gibt zwei Typen von Kubernetes-Arbeitslastclustern, die in der vSphere IaaS control plane-Infrastruktur bereitgestellt werden können, in der Supervisor als Verwaltungscluster fungiert und die Kubernetes-Cluster-API (CAPI) hostet. Jeder Typ basiert auf der ClusterClass. Informationen zu den unterstützten Versionen finden Sie in den Versionshinweisen zu TKR. Siehe auch Verwenden von Kubernetes-Versionen mit TKG-Dienstclustern

TanzuKubernetesCluster mit CAPI-Cluster bezieht sich auf die standardmäßige ClusterClass mit dem Namen „tanzukubernetescluster“

Cluster-Signatur:

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
```

Diese Art von Arbeitslastcluster ist ein Tanzu Kubernetes-Cluster, der mithilfe der Cluster-API (CAPI) erstellt wurde und sich auf eine standardmäßige ClusterClass mit dem Namen `tanzukubernetescluster` bezieht. Der Clustertyp lautet **TanzuKubernetesCluster** und die API für die Bereitstellung lautet `v1alpha3`. Da es sich hierbei um eine Abstraktion über einem CAPI-Cluster handelt, wird der Verweis auf die Backend-Cluster-Klasse im Cluster-Manifest nicht angegeben. Die Referenz wird vom System verarbeitet.

Bei dieser Art von Arbeitslastcluster ist das TanzuKubernetesCluster-Objekt an vorderster Stelle und dient als Abstraktionsschicht. Es gibt für die Bereitstellung dieses Clustertyps keine Änderung im Workflow im Vergleich zur Bereitstellung eines TKGS-Clusters auf vSphere IaaS control plane der Version 7.

CAPI-Cluster bezieht sich auf die standardmäßige ClusterClass mit dem Namen „tanzukubernetescluster“

Cluster-Signatur:

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
```

Diese Art von Arbeitslastcluster ist ein reiner CAPI-Cluster, der mithilfe der Cluster-API erstellt wurde und sich auf eine standardmäßige ClusterClass mit dem Namen `tanzukubernetescluster` bezieht. Der Clustertyp lautet **Cluster** und die API für die Bereitstellung lautet `v1beta1`.

Bei diesem Typ von Arbeitslastcluster ist die CAPI-Cluster-API an vorderster Stelle. Es gibt keine TKC-Abstraktionsschicht. Das System stellt einen Controller für die Handhabung der Infrastruktur bereit, sodass Sie keine ClusterClass mit Bezug auf Objekte erstellen müssen. Variablen werden angezeigt, sodass Sie den Cluster anpassen können. Die Felder in der Clusterspezifikation unterscheiden sich von den Feldern in der TKC-Spezifikation, aber der Bereitstellungsworkflow ist identisch.

APIs für die TKG-Cluster-Bereitstellung

TKG auf vSphere 8 Supervisor bietet zwei APIs für die Verwaltung des Lebenszyklus von TKG-Clustern: `v1alpha3` und `v1beta1`. Beide APIs sind naturgemäß deklarativ, ähnlich wie die Kubernetes-API. Bei deklarativer Clusterbereitstellung geben Sie den gewünschten Zustand des TKG-Clusters an: Anzahl der Knoten, verfügbarer Speicher, VM-Größen, Kubernetes-Softwareversion. TKG übernimmt die Aufgabe der Bereitstellung und Pflege eines Clusters, der dem deklarierten Zustand entspricht.

Wenn Sie ein Upgrade eines vorhandenen Tanzu Kubernetes-Clusters auf TKG auf vSphere 8 Supervisor durchführen, muss dieser Cluster die `v1alpha2`-API verwenden, bevor Sie mit dem Upgrade-Vorgang beginnen. Vollständige Informationen finden Sie in der Dokumentation zum Upgrade: #unique_51.

API	Art	vCenter-Version	Beschreibung
v1beta1	Cluster	vCenter 8+	Neue API zur Verwaltung des Lebenszyklus eines Clusters basierend auf einer Clusterklasse.
v1alpha3	TanzuKubernetesCluster	vCenter 8+	Fortführung der v1alpha2-API. Alle von der API v1alpha2 unterstützten Funktionen werden von der v1alpha3-API unterstützt. Neue Funktionen hinzugefügt.

API	Art	vCenter-Version	Beschreibung
v1alpha2	TanzuKubernetesCluster	vCenter 7 U3	Legacy-API für die Bereitstellung von Tanzu Kubernetes-Clustern auf vCenter 7 U3 Supervisor und für das Upgrade von Clustern auf vCenter 8 Supervisor. Beim Upgrade oder bei der Bereitstellung auf vSphere 8 wird die v1alpha2-API automatisch in die v1alpha3-API umgewandelt.
v1alpha1	TanzuKubernetesCluster	vCenter 7 U1, U2	Veraltete API für die Bereitstellung von Tanzu Kubernetes-Clustern auf der ersten Generation von vCenter 7 Supervisor.

Clients für die TKG-Clusterbereitstellung

TKG auf vSphere 8 Supervisor unterstützt verschiedene Client-Workflows für die Bereitstellung von TKG-Clustern:

- Kubectl + YAML für deklarative Clusterbereitstellung im Kubernetes-Stil. Weitere Informationen finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl.
- Tanzu-CLI für die interaktive Bereitstellung von Clustern über die Befehlszeile. Weitere Informationen finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe der Tanzu-CLI.
- Tanzu Mission Control für webbasierte Clusterbereitstellung. Weitere Informationen finden Sie unter Registrieren von Tanzu Mission Control, die bei Supervisor gehostet werden.

Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl

Verwenden Sie diesen Workflow, um einen TKG-Dienstcluster deklarativ mithilfe von kubectl-Befehlen und einer in der YAML definierten Clusterspezifikation bereitzustellen.

Dieser Workflow unterstützt die deklarative Bereitstellung eines TKG-Clusters mithilfe von kubectl und YAML.

Voraussetzungen

Überprüfen oder erfüllen Sie die folgenden Voraussetzungen, bevor Sie den Bereitstellungs-Workflow starten:

- Installieren Sie die neueste Supervisor-Version oder aktualisieren Sie Ihre Umgebung auf diese Version. Weitere Informationen hierzu finden Sie unter Kapitel 2 Ausführen von TKG-Dienstclustern.

- Erstellen oder aktualisieren Sie eine Inhaltsbibliothek mit den neuesten Tanzu Kubernetes-Versionen. Weitere Informationen hierzu finden Sie unter Kapitel 5 Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster.
- Erstellen und konfigurieren Sie einen vSphere-Namespace für das Hosting von TKG-Clustern. Weitere Informationen hierzu finden Sie unter Kapitel 6 Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern.

Verfahren

1 Installieren Sie den Kubernetes-CLI-Tools für vSphere.

  Installieren des Kubernetes-CLI-Tools für vSphere.

2 Authentifizieren Sie sich mithilfe von kubectl bei Supervisor.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-
username USERNAME
```

  Hinweis FQDN für Supervisor kann nur verwendet werden, wenn er aktiviert ist. In der Dokumentation zur Supervisor-Konfiguration finden Sie ausführliche Informationen.

3 Überprüfen Sie, ob die Anmeldung beim Supervisor erfolgreich war.

  Sinngemäß sollte die folgende Meldung angezeigt werden:

```
Logged in successfully.

You have access to the following contexts:
   192.197.2.65
   tkg2-cluster-namespace
```

  Dabei ist `192.197.2.65` der Supervisor-Kontext und `tkg2-cluster-namespace` der Kontext für den vSphere-Namespace, in dem Sie den TKG-Cluster bereitstellen möchten.

4 Stellen Sie sicher, dass der Ziel-vSphere-Namespace der aktuelle Kontext ist.

```
kubectl config get-contexts
```

```
CURRENT   NAME                      CLUSTER
AUTHINFO                                        NAMESPACE
          192.197.2.65              192.197.2.65
wcp:10.197.154.65:administrator@vsphere.local
*         tkg2-cluster-namespace    10.197.154.65
wcp:10.197.154.65:administrator@vsphere.local   tkg2-cluster-namespace
```

  Wenn der Ziel-vSphere-Namespace nicht der aktuelle Kontext ist, wechseln Sie zu diesem.

```
kubectl config use-context tkg2-cluster-namespace
```

5 Listen Sie die VM-Klassen auf, die im vSphere-Namespace verfügbar sind.

```
kubectl get virtualmachineclass
```

Sie können nur VM-Klassen verwenden, die an den Ziel-Namespace gebunden sind. Wenn keine VM-Klassen angezeigt werden, stellen Sie sicher, dass die VM-Standardklassen mit dem vSphere-Namespace verknüpft wurden. Siehe auch Beheben von Fehlern bei VM-Klassen

6 Rufen Sie die verfügbaren Klassen persistenter Speicher-Volumes ab.

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

Der Befehl gibt Details zum Namespace zurück, einschließlich der Speicherklasse im Format `tkg2-storage-policy.storageclass.storage.k8s.io/requests.storage`. Das erste Token der Zeichenfolge ist der Name der Speicherklasse, in diesem Beispiel `tkg2-storage-policy`. Der Befehl `kubectl describe storageclasses` gibt auch verfügbare Speicherklassen zurück, erfordert jedoch vSphere-Administratorberechtigungen.

7 Listen Sie die verfügbaren Tanzu Kubernetes-Versionen auf.

Sie können einen der folgenden Befehle verwenden, um diesen Vorgang auszuführen:

```
kubectl get tkr
```

```
kubectl get tanzukubernetesreleases
```

Dieser Befehl gibt die TKRs zurück, die in diesem vSphere-Namespace zur Verfügung stehen, und zeigt deren Kompatibilität mit dem Supervisor an, auf dem Sie die Bereitstellung durchführen. Sie können nur die Versionen verwenden, die von diesem Befehl zurückgegeben werden. Wenn Sie keine oder nicht die gewünschten Versionen sehen, stellen Sie sicher, dass Sie folgende Schritte ausgeführt haben: a) eine TKR-Inhaltsbibliothek erstellt; b) die Inhaltsbibliothek mit den gewünschten OVA-Dateien synchronisiert; und c) die Inhaltsbibliothek mit dem vSphere-Namespace verknüpft, in dem Sie den TKG-Cluster bereitstellen.

8 Erstellen Sie die YAML-Datei für die Bereitstellung des TKG-Clusters.

a Ermitteln Sie den Typ des zu erstellenden Clusters und überprüfen Sie dessen API und Funktionen:

- TanzuKubernetesCluster: Verwenden der v1alpha3-API von TanzuKubernetesCluster
- Cluster: Verwenden der v1beta1-API von Clustern

b Beginnen Sie mit einer der Beispiel-YAMLs für die Bereitstellung des Clusters.

Beispiel:

- v1alpha3-Beispiel: Standard-TanzuKubernetesCluster
- v1beta1-Beispiel: Standardcluster

c Speichern Sie die YAML-Datei als `tkg2-cluster-1.yaml` oder ähnlich.

d Füllen Sie die YAML mit den Informationen auf, die Sie aus der Ausgabe der vorherigen Befehle erhalten haben. Dazu zählen:

- Des Namens des Clusters, wie z. B. `tkg2-cluster-1`
- De zielseitige vSphere-Namespace
- Gebundener VM-Klassen, wie z. B. `guaranteed-medium`
- Speicherklasse für Clusterknoten und persistente Volumes
- Der Anzahl der Steuerungsebenen- und Worker-Knoten (Replikate)
- Der von der TKR-NAME-Zeichenfolge angegebenen Tanzu Kubernetes-Version, wie z. B. `v1.25.7+vmware.3-fips.1-tkg.1`

e Passen Sie die YAML des TGK-Clusters nach Bedarf an.

- Hinzufügen eines separaten Volumes für Komponenten mit hoher Änderungsrate, wie z. B. `containerd`
- Festlegen einer Standardklasse für dauerhaften Speicher für Clusterknoten und dauerhafte Volumes
- Anpassen des Clusternetzwerks, einschließlich CNI, Pod und Dienst-CIDRs

f Verwenden Sie eine YAML-Syntaxprüfung und stellen Sie sicher, dass die YAML gültig ist.

Das Ergebnis dieses Schritts ist eine gültige YAML für die Bereitstellung des TKG-Clusters.

9 Stellen Sie den TKG-Cluster bereit und führen Sie den folgenden Befehl aus.

```
kubectl apply -f tkg2-cluster-1.yaml
```

Erwartetes Ergebnis:

```
tanzukubernetescluster.run.tanzu.vmware.com/tkg2-cluster-1 created
```

10 Überwachen Sie die Bereitstellung des TKG-Clusters.

```
kubectl get tanzukubernetesclusters
```

```
kubectl get tkc
```

Oder Sie erstellen einen Cluster mit der v1beta1-API:

```
kubectl get cluster
```

Anfänglich lautet der Bereitschaftsstatus (READY) „False“, da der Cluster bereitgestellt wird. Nach einigen Minuten sollte der Bereitschaftsstatus „True“ lauten.

```
NAME              CONTROL PLANE   WORKER   TKR NAME                              AGE
READY   TKR COMPATIBLE   UPDATES AVAILABLE
tkg2-cluster-1    3               6        v1.25.7+vmware.3-fips.1-tkg.1       49m
True    True
```

Führen Sie zusätzliche Befehle aus, um Details zum Cluster anzuzeigen.

```
kubectl get
tanzukubernetescluster,cluster,virtualmachinesetresourcepolicy,virtualmachineservice,kubead
mcontrolplane,machinedeployment,machine,virtualmachine
```

```
kubectl describe tanzukubernetescluster tkg2-cluster-1
```

11 Überwachen Sie die Bereitstellung der Clusterknoten mithilfe des vSphere Client.

In der vSphere-Bestandsliste für **Hosts und Cluster** sollte zu sehen sein, dass die Clusterknoten-VMs im zielseitigen vSphere-Namespace bereitgestellt werden.

12 Sobald sich alle TKG-Clusterknoten im Zustand BEREIT befinden, melden Sie sich mit dem vSphere-Plug-In für kubectl beim Cluster an.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN \
--vsphere-username USERNAME \
--tanzu-kubernetes-cluster-name CLUSTER-NAME \
--tanzu-kubernetes-cluster-namespace NAMESPACE-NAME
```

Beispiel:

```
kubectl vsphere login --server=192.197.2.65 --vsphere-username user@vsphere.local \
--tanzu-kubernetes-cluster-name tkg2-cluster-1 --tanzu-kubernetes-cluster-namespace tkg2-
cluster-namespace
```

Hinweis Eine erfolgreiche Ausführung des Anmeldebefehls findet erst statt, wenn die Steuerungsebenenknoten ausgeführt werden und das Authentifizierungsdienst-Plug-In gestartet wurde. Während der Erstellung der Worker-Knoten verläuft die Anmeldung unter Umständen instabil. Es wird empfohlen, dass Sie sich anmelden, sobald sich alle Clusterknoten im Zustand BEREIT befinden.

13 Wechseln Sie den Kontext zum TKG-Cluster, sodass dieser der aktuelle Kontext ist.

Nach erfolgreicher Anmeldung beim TKG-Cluster sollte eine Meldung ähnlich der folgenden angezeigt werden.

```
Logged in successfully.
```

```
You have access to the following contexts:
   192.197.2.65
   tkg2-cluster-namespace
   tkg2-cluster-1
```

Dabei ist `192.197.2.65` der Supervisor-Kontext, `tkg2-cluster-namespace` der vSphere-Namespace-Kontext und `tkg2-cluster-1` der TKG-Clusterkontext.

Wechseln Sie zum TKG-Clusterkontext.

```
kubect config use-context tkg2-cluster-1
```

14 Überprüfen Sie die TKG-Clusterressourcen.

```
kubectl get nodes
```

```
kubectl get namespaces
```

```
kubectl get pods -A
```

```
kubectl cluster-info
```

```
kubectl api-resources
```

15 Testen Sie den TKG-Cluster, indem Sie einen Test-Pod bereitstellen und überprüfen Sie, ob er wie erwartet funktioniert.

Weitere Informationen finden Sie unter Testen der TKG-Clusterbereitstellung mithilfe von kubectl.

Workflow zum Bereitstellen von TKG-Clustern auf mithilfe der Tanzu-CLI

Führen Sie diesen Workflow aus, um einen v1beta1-TKG-Cluster mithilfe der Tanzu-CLI bereitzustellen.

Voraussetzungen

Überprüfen oder erfüllen Sie die folgenden Voraussetzungen, bevor Sie den Bereitstellungs-Workflow starten:

- Installieren Sie die neueste Supervisor-Version oder aktualisieren Sie Ihre Umgebung auf diese Version. Weitere Informationen hierzu finden Sie unter Kapitel 2 Ausführen von TKG-Dienstclustern.
- Erstellen oder aktualisieren Sie eine Inhaltsbibliothek mit den neuesten Tanzu Kubernetes-Versionen. Weitere Informationen hierzu finden Sie unter Kapitel 5 Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster.

- Erstellen und konfigurieren Sie einen vSphere-Namespace für das Hosting von TKG 2.0-Clustern. Weitere Informationen hierzu finden Sie unter Kapitel 6 Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern.

Bereitstellen eines Standard-TKG-Clusters

Führen Sie die folgenden Schritte aus, um einen standardmäßigen v1beta1-Cluster mithilfe der Tanzu-CLI bereitzustellen. Weitere Anleitungen oder Fehlerbehebungen finden Sie unter Erstellen von Arbeitslastclustern in der eigenständigen TKG-Dokumentation.

1 Installieren Sie die Tanzu-CLI.

  Weitere Informationen finden Sie unter Installieren der Tanzu-CLI zur Verwendung mit TKG-Dienst-Clustern.

2 Stellen Sie mithilfe der Tanzu-CLI eine Verbindung zu Supervisor her.

  - Herstellen einer Verbindung zu Supervisor mithilfe der Tanzu-CLI und der vCenter SSO-Authentifizierung oder
  - Herstellen einer Verbindung zu Supervisor mit der Tanzu-CLI und einem externen IDP

3 Listen Sie die verfügbaren TKrs auf.

```
tanzu kubernetes-release get
```

4 Erstellen Sie ein Clustermanifest mit den gewünschten Konfigurationen.

  Mit TKG auf vSphere 8 Supervisor können Sie eine Objektspezifikation im Kubernetes-Stil mit der Tanzu-CLI verwenden, um einen Cluster basierend auf einer ClusterClass zu erstellen.

  a Beginnen Sie mit dem v1beta1-Beispiel: Standardcluster.

  b Füllen Sie `spec.clusterNetwork` mit den erforderlichen `cidrBlocks` auf.

  c Füllen Sie `spec.topology` mit den in der Tabelle aufgelisteten erwarteten Werten auf.

    - TKR NAME-Zeichenfolge, wie z. B. `v1.26.13---vmware.1-fips.1-tkg.3`
    - Anzahl der Steuerungsebenenknoten, z. B. `3`
    - Name jedes Worker-Knotenpools, z. B. `node-pool-1`
    - Anzahl der Worker-Knoten, z. B. `3`
    - VM-Klasse, z. B. `guaranteed-medium`
    - Speicherklasse, z. B. `tkg2-storage-policy`

5 Speichern Sie das Clustermanifest als `cluster-default.yaml` und validieren Sie es mithilfe einer YAML-Prüfung.

6 Erstellen Sie den TKG-Cluster.

```
tanzu cluster create -f cluster-default.yaml
```

Hinweis Hängen Sie `-v 8` für die ausführliche Ausgabe an.

7 Stellen Sie sicher, dass der TKG-Cluster erstellt wird.

```
Workload cluster 'cluster-default' created
```

8 Nachdem der Cluster erstellt wurde, führen Sie den folgenden Befehl aus, um den Status des Clusters zu überprüfen.

```
tanzu cluster get cluster-default
```

9 Listen Sie den Cluster auf.

```
tanzu cluster list
```

10 Überprüfen Sie die Clusterknoten.

```
tanzu cluster machinehealthcheck node get cluster-default
```

```
tanzu cluster machinehealthcheck control-plane get cluster-default
```

11 Rufen Sie den Konfigurationskontext für den TKG-Cluster ab.

```
tanzu cluster kubeconfig get cluster-default -n tkg2-cluster-ns
```

12 Greifen Sie auf den Cluster zu.

```
kubectl config use-context tanzu-cli-cluster-default@cluster-default
```

13 Testen Sie den TKG 2.0-Cluster, indem Sie einen Test-Pod bereitstellen, und überprüfen Sie, ob er wie erwartet funktioniert.

Weitere Informationen finden Sie unter Testen der TKG-Clusterbereitstellung mithilfe von kubectl.

Bereitstellen eines benutzerdefinierten TKG-Clusters auf Supervisor

Um einen benutzerdefinierten v1beta1-Cluster wie v1beta1-Beispiel: Cluster mit Calico-CNI bereitzustellen, können Sie alle Spezifikationen wie im Beispiel angegeben in eine einzelne YAML einfügen, bestimmte Werte ändern, sodass sie Ihrer Umgebung entsprechen, und beispielsweise `kubectl apply -f cluster-calico.yaml` ausführen.

Wenn Sie denselben benutzerdefinierten v1beta1-Cluster mithilfe der Tanzu-CLI bereitstellen möchten, sollten die Konfigurationsobjekte `CalicoConfig` und `ClusterBootstrap` vorhanden sein, bevor Sie den Cluster erstellen.

So stellen Sie den Cluster mit der Calico-CNI bereit:

1 Erstellen Sie die YAML für die Konfigurationsobjekte `CalicoConfig` und `ClusterBootstrap` jeweils mit dem gewünschten Clusternamen und Namespace.

2 Führen Sie `kubectl apply -f` für jedes der drei Konfigurationsobjekte aus oder legen Sie sie in einer einzelnen YAML ab und führen Sie `kubectl apply -f` aus.

3 Erstellen Sie die Clusterspezifikation `cluster-calico.yaml` mit dem Namen und dem Namespace, die denen in den Konfigurationsobjekten entsprechen, und allen anderen gewünschten Parametern.

4 Erstellen Sie den Cluster.

```
tanzu cluster create -f cluster-calico.yaml
```

Testen der TKG-Clusterbereitstellung mithilfe von kubectl

Nachdem Sie einen TKG-Cluster bereitgestellt haben, wird empfohlen, eine Testarbeitslast bereitzustellen und die Cluster-Funktionalität zu testen.

Stellen Sie eine Testanwendung bereit, um zu überprüfen, ob Ihr TKG-Cluster betriebsbereit ist und ausgeführt wird.

Voraussetzungen

- Stellen Sie einen TKG-Cluster bereit.
- Stellen Sie eine Verbindung mit dem TKG-Cluster her.

Verfahren

1 Stellen Sie einen TKG-Cluster bereit.

Weitere Informationen hierzu finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl.

2 Melden Sie sich mithilfe von kubectl bei Supervisor an.

```
kubectl vsphere login --server=<IP or FQDN> --vsphere-username <USERNAME>
```

3 Ändern Sie den Konfigurationskontext in den vSphere-Namespace, in dem der TKG-Cluster bereitgestellt wird.

```
kubectl config use-context VSPHERE-NAMESPACE
```

4 Melden Sie sich beim TKG-Zielcluster an.

```
kubectl vsphere login --server=<IP or FQDN> --vsphere-username <USERNAME> \
--tanzu-kubernetes-cluster-name CLUSTER-NAME \
--tanzu-kubernetes-cluster-namespace NAMESPACE-NAME
```

5 Erstellen Sie die Datei `ping-pod.yaml` mit folgendem Inhalt.

```
apiVersion: v1
kind: Pod
metadata:
  name: ping-pod
  namespace: default
spec:
  containers:
  - image: busybox:1.34
    name: busybox
    command: ["ping", "-c"]
    args: ["1", "8.8.8.8"]
  imagePullSecrets:
  - name: regcred
  restartPolicy: Never
```

6 Erstellen Sie die Anmeldedaten für die `regcred`-Registrierung.

Das für dieses Szenario verwendete Container-Image (busybox) wird aus der öffentlichen Docker Hub-Registrierung abgerufen, wodurch das Abrufen von Images eingeschränkt werden kann. Wenn dies der Fall ist, benötigen Sie ein Docker Hub-Konto und einen geheimen Schlüssel zum Abrufen von Images („regcred“), auf den in der Pod-Spezifikation verwiesen wird. Informationen zum Erstellen dieses geheimen Schlüssels finden Sie unter Erstellen eines geheimen Schlüssels für die private Registrierung.

7 Konfigurieren Sie gegebenenfalls Pod-Sicherheit.

Wenn Sie Tanzu Kubernetes Release v1.24 oder früher verwenden, fahren Sie mit dem nächsten Schritt fort und erstellen Sie den Pod.

Wenn Sie Tanzu Kubernetes Release v1.25 verwenden, sind Konfigurieren von PSA für TKR 1.25 und höher aktiviert. Sie können mit dem nächsten Schritt fortfahren und den Pod erstellen. Beachten Sie jedoch, dass Sie eine Warnung bezüglich Pod-Sicherheitsverstößen erhalten, die Sie ignorieren können.

```
Warning: would violate PodSecurity "restricted:latest": allowPrivilegeEscalation != false
(container "busybox" must set securityContext.allowPrivilegeEscalation=false),
unrestricted capabilities (container "busybox" must set
securityContext.capabilities.drop=["ALL"]),
runAsNonRoot != true (pod or container "busybox" must set
securityContext.runAsNonRoot=true),
seccompProfile (pod or container "busybox" must set securityContext.seccompProfile.type to
"RuntimeDefault" or "Localhost")
```

Wenn Sie Tanzu Kubernetes Release v1.26 oder höher verwenden, werden Konfigurieren von PSA für TKR 1.25 und höher erzwungen. Wenn Sie den Pod wie im nächsten Schritt gezeigt erstellen, tritt folgender Fehler auf.

```
Error from server (Forbidden): error when creating "ping-pod.yaml": pods "ping-pod" is
forbidden:
violates PodSecurity "restricted:latest": allowPrivilegeEscalation != false
(container "busybox" must set securityContext.allowPrivilegeEscalation=false),
```

```
unrestricted capabilities (container "busybox" must set
securityContext.capabilities.drop=["ALL"]),
runAsNonRoot != true (pod or container "busybox" must set
securityContext.runAsNonRoot=true),
seccompProfile (pod or container "busybox" must set securityContext.seccompProfile.type to
"RuntimeDefault" or "Localhost")
```

Zur Fehlerbehebung führen Sie folgenden Befehl in dem `default`-Namespace aus, in dem der Pod erstellt wird. Beachten Sie, dass Sie dadurch Konfigurieren von PSA für TKR 1.25 und höher im Namespace entfernen.

```
kubectl label --overwrite ns default pod-security.kubernetes.io/enforce=privileged
```

Alternativ können Sie securityContext direkt auf den Pod anwenden, wie z. B.:

```
...
spec:
  containers:
  - image: busybox:1.34
    name: busybox
    command: ["ping", "-c"]
    args: ["1", "8.8.8.8"]
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
      runAsNonRoot: true
      runAsUser: 1000
      seccompProfile:
        type: "RuntimeDefault"
...
```

8 Wenden Sie die YAML an.

```
kubectl apply -f ping-pod.yaml
```

Erwartetes Ergebnis:

```
pod/ping-pod created
```

9 Überprüfen Sie, ob der Pod erfolgreich abgeschlossen wurde.

```
kubectl get pods -n default
```

```
NAME       READY   STATUS      RESTARTS   AGE
ping-pod   0/1     Completed   0          13s
```

10 Stellen Sie sicher, dass der Pod den DNS-Server angepingt hat.

```
kubectl logs ping-pod -f
```

Erwartetes Ergebnis:

```
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=106 time=33.352 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 33.352/33.352/33.352 ms
```

11 Löschen Sie den Pod.

```
kubectl delete -f ping-pod.yaml
```

12 Überprüfen Sie, ob der Pod gelöscht wurde.

```
kubectl get pods
```

Löschen eines TKG-Clusters mithilfe von Kubectl oder der Tanzu-CLI

Verwenden Sie `kubectl` oder die Tanzu-CLI, um einen TKG-Cluster zu löschen.

Wenn Sie einen Tanzu Kubernetes-Cluster mit `kubectl` oder der Tanzu-CLI löschen, stellt Kubernetes Garbage Collection sicher, dass alle abhängigen Ressourcen gelöscht werden.

Hinweis Versuchen Sie nicht, einen Cluster über die vSphere Client- oder vCenter Server-CLI zu löschen.

Verfahren

1 Authentifizieren Sie sich beim Supervisor.

2 Ändern Sie den Kontext in den vSphere-Namespace, in dem die zu löschende TKG -Instanz bereitgestellt wird.

```
kubectl config use-context CLUSTER-NAMESPACE
```

3 Listet die TKG-Cluster im aktuellen Namespace auf.

```
kubectl get clusters
```

```
tanzu cluster list
```

4 Löschen Sie den TKG-Cluster mit der folgenden Syntax.

v1alpha3-Cluster mit kubectl:

```
kubectl delete tanzukubernetescluster --namespace CLUSTER-NAMESPACE CLUSTER-NAME
```

v1beta1-Cluster mit kubectl:

```
kubectl delete cluster --namespace CLUSTER-NAMESPACE CLUSTER-NAME
```

v1alpha3- oder v1beta1-Cluster mit der Tanzu-CLI:

```
tanzu cluster delete --namespace CLUSTER-NAMESPACE CLUSTER-NAME
```

Beispielergebnis:

```
tanzukubernetescluster.run.tanzu.vmware.com "tkg-cluster-1" deleted
```

5 Überprüfen Sie, ob der Cluster gelöscht wurde.

kubectl:

```
kubectl get clusters
```

Tanzu-CLI:

```
tanzu cluster list
```

6 Löschen Sie den Clusterkontext aus der `kubeconfig`-Datei.

```
kubectl config delete-context CONTEXT
```

Beispiel:

```
kubectl config get-contexts
CURRENT   NAME            CLUSTER         AUTHINFO
NAMESPACE
          192.0.2.1       192.0.2.1       wcp:192.0.2.1:administrator@vsphere.local
          tkg-cluster-1   192.0.2.6       wcp:192.0.2.6:administrator@vsphere.local
*         tkg-ns-1        192.0.2.7       wcp:192.0.2.7:administrator@vsphere.local
tkg-ns-1
```

```
kubectl config delete-context tkg-cluster-1
deleted context tkg-cluster-1 from $HOME/.kube/config
```

```
kubectl config get-contexts
CURRENT   NAME            CLUSTER        AUTHINFO
NAMESPACE
          192.0.2.1       192.0.2.1      wcp:192.0.2.1:administrator@vsphere.local
*         tkg-ns-1        192.0.2.7       wcp:192.0.2.7:administrator@vsphere.local
tkg-ns-1
```

Verwenden der v1beta1-API von Clustern

Dieser Abschnitt enthält Referenzinhalte für die Bereitstellung eines Clusters mithilfe der v1beta1-API, einschließlich Beispielen mit verschiedenen Konfigurationen und Anpassungen, die Ihren Anforderungen entsprechen.

Cluster-API v1beta1

Mit der Cluster-API v1beta1 können Sie einen Cluster basierend auf einer standardmäßigen ClusterClass-Definition bereitstellen.

ClusterClass API v1beta1

Die Kubernetes Cluster-API ist eine Suite von Tools, die die deklarative Bereitstellung, Upgrades und den Betrieb von Kubernetes-Clustern ermöglichen. ClusterClass ist eine Weiterentwicklung der Cluster-API, mit der Sie Vorlagen zur Verwaltung des Lebenszyklus von Clustergruppen definieren können. TKG-Dienst unterstützt ClusterClass über die v1beta1-API.

TKG-Dienst wird mit einer standardmäßigen ClusterClass-Definition mit dem Namen `tanzukubernetescluster` ausgeliefert. Die ClusterClass `tanzukubernetescluster` stellt die Vorlage für die Erstellung von Clustern mithilfe der v1beta-API bereit. Die ClusterClass `tanzukubernetescluster` ist in allen Benutzer-Namespaces verfügbar. Um einen Cluster auf Basis dieser ClusterClass zu erstellen, verweisen Sie einfach in der Clusterspezifikation auf ihn. Weitere Informationen finden Sie in den v1beta-Beispielen.

Standard-ClusterClass tanzukubernetescluster

Die standardmäßige ClusterClass `tanzukubernetescluster` ist unveränderlich. Sie kann unter Umständen mit jeder Version des TKG-Diensts aktualisiert werden.

Führen Sie die folgenden Schritte durch, um die Standard-ClusterClass `tanzukubernetescluster` anzuzeigen, die im Lieferumfang der TKG-Dienstinstanz enthalten ist:

1 Melden Sie sich bei Supervisor an.

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USER@vsphere.local
```

2 Ändern Sie den Kontext in den vSphere-Namespace, in dem ein TKG-Cluster bereitgestellt wird.

```
kubeclt config use-context VSPEHRE-NS
```

3 Rufen Sie die Standard-ClusterClass `tanzukubernetescluster` ab.

```
kubectl get clusterclass tanzukubernetescluster -o yaml
```

4 Sie können die Ausgabe der Standard-ClusterClass optional in eine Datei mit der Bezeichnung *tkc-dcc.yaml* schreiben.

```
kubectl get clusterclass tanzukubernetescluster -o yaml > tkc-dcc.yaml
```

ClusterClass-Variablen zum Anpassen eines Clusters

Sie passen einen Cluster basierend auf der ClusterClass `tanzukubernetescluster` mithilfe von Variablen an. Variablen werden mithilfe von Name-Wert-Paaren definiert. Die Syntax muss dem openAPIV3Schema entsprechen.

Für die Bereitstellung eines Clusters mithilfe der v1beta1-API sind zwei Variablen erforderlich:

- VM-Klasse
- Speicherklasse

Zusätzliche Variablen sind für die Anpassung eines Clusters verfügbar, wie z. B.:

- Proxy
- TLS-Zertifikate
- SSH-Schlüssel

In den folgenden Abschnitten werden alle Variablen aufgelistet, die mit der Standard-ClusterClass `tanzukubernetescluster` verfügbar sind.

Wichtig Ein gültiger Schlüsselname darf nur aus alphanumerischen Zeichen, einem Bindestrich (z. B. `key-name`), einem Unterstrich (z. B. `KEY_NAME`) oder einem Punkt (z. B. `key.name`) bestehen. Leerzeichen können in Schlüsselnamen nicht verwendet werden.

clusterEncryptionConfigYaml

Verwenden Sie die `clusterEncryptionConfigYaml`-Variable, um die Clusterverschlüsselung zu konfigurieren.

clusterEncryptionConfigYaml

Zeichenfolge, bei der es sich um eine YAML-Datei mit Details zur Verschlüsselungskonfiguration handelt.

Sie können die Verschlüsselung von Daten in der etcd-Datenbank mit dem Paket kube-apiserver Encryption Configuration konfigurieren. Weitere Informationen finden Sie unter Verschlüsseln von geheimen inaktiven Daten in der Kubernetes-Dokumentation.

```
...
    variables:
    #clusterEncryptionConfigYaml specifies the base64 encoded
    #EncryptionConfiguration YAML
    #the YAML contains a base64 encryption configuration for the cluster identity
    #the key is generated randomly
```

```
    - name: clusterEncryptionConfigYaml
      value: string which is name of the EncryptionConfiguration YAML
```

controlPlaneCertificateRotation

Verwenden Sie die `controlPlaneCertificateRotation`-Variable, um das System so zu konfigurieren, dass die TLS-Zertifikate für Knoten der Steuerungsebene rotiert werden, indem ein Rollout dieser Zertifikate vor deren Ablauf ausgelöst wird. Die Zertifikatrotation der Steuerungsebene ist für alle neuen und vorhandenen Knoten der Steuerungsebene verfügbar.

controlPlaneCertificateRotation

Boolesche Variable zur Aktivierung der Funktion und Anzahl der Tage vor Ablauf für die Rotation der Zertifikate. Weitere Informationen finden Sie unter Automatically rotating certificates using Kubeadm Control Plane provider (Automatische Rotation von Zertifikaten mithilfe des Kubeadm-Steuerungsebenenanbieters).

```
...
    variables:
    - name: controlPlaneCertificateRotation
      value:
        activate: true
        daysBefore: 90
```

Dabei gilt:

- `activate` ist ein boolescher Wert für die Aktivierung der Funktion. Der Standardwert lautet `true`.
- `daysBefore` ist die Dauer in der Anzahl der Tage vor Ablauf. Der Standardwert beträgt `90` Tage. Der Mindestwert beläuft sich auf `7` Tage vor Ablauf.

Hinweis Diese Variable wurde für vSphere 8 Update 3 (TKG-Dienst 3.0) aktualisiert.

controlPlaneVolumes

Verwenden Sie die `controlPlaneVolumes`-Variable, um dauerhafte Volumes für Steuerungsebenenknoten zu konfigurieren.

controlPlaneVolumes

Optionales Array von Objekten, von denen jedes `name`, `storageClass` und `mountPath` enthält, die jeweils Zeichenfolgen sind, und ein optionales `capacity`-Objekt, das eine `storage`-Zeichenfolge enthält.

```
...
    variables:
      #controlPlaneVolumes is an optional set of PVCs to create and
      #attach to each node
      - name: controlPlaneVolumes
        value:
          #name of the PVC to be used as the suffix (node.name)
```

```
      - name: NAME
        #mountPath is the directory where the volume device is mounted
        #takes the form /dir/path
        mountPath: /dir/path
        #storageClass is the storage class to use for the PVC
        storageClass: tkgs-storage-profile
        #capacity is the PVC storage capacity
        capacity:
          #storage sets the capacity for the disk volume
          #if not specified defaults to storageClass capacity
          storage: 4Gi
```

defaultRegistrySecret

Mit der Variablen `defaultRegistrySecret` wird die Standard-Containerregistrierung für den Cluster konfiguriert.

Hinweis Diese Variable ist für die Verwendung mit der eingebetteten Harbor-Registrierung reserviert.

defaultRegistrySecret

Objekt, das einen öffentlichen Schlüssel, einen Zertifikatsnamen und einen Namespace für die Standard-Containerregistrierung enthält.

Wenn Sie die eingebettete Harbor-Registrierung auf Supervisor aktivieren, gibt die Variable `defaultRegistrySecret` das Registrierungsdienstzertifikat an, dem der Cluster vertraut. Der geheime Zertifikatschlüssel ist mit `managed-by: vmware-vRegistry` gekennzeichnet. Bei der Clustererstellung wird ein defaultRegistry-Zertifikat in die Variable `defaultRegistrySecret` eingefügt. Nach der Clustererstellung verwalten Sie die Zertifikatrotation oder ein beliebiges Update, indem Sie die Variable manuell aktualisieren.

```
...
    variables:
    - name: defaultRegistrySecret
      value:
        #data holds the base64 encoded data.ca\.crt content
        #data.ca\.crt is already encoded, so raw cert data is encoded twice
        data: LS0tLS1CRUdJTiBDRVJU...S0tRU5EIENFUlRJRklDQVRFL
        #name specifies the name of the registry cert secret
        name: harbor-ca-key-pair
        #namespace specifies the ns of the registry cert secret
        namespace: svc-harbor-domain-c9
```

defaultStorageClass

Verwenden Sie die `defaultStorageClass`-Variable, um eine Standardspeicherklasse für den Cluster zu konfigurieren.

defaultStorageClass

Zeichenfolge, die angibt, welche Speicherklasse als Standardspeicherklasse verwendet werden soll, die häufig von bestimmten Anwendungen wie Helm-Diagrammen und Tanzu-Paketen benötigt wird.

```
...
    variables:
    - name: defaultStorageClass
      value: tkg2-storage-profile
```

extensionCert

Verwenden Sie die `extensionCert`-Variable, um ein TLS-Zertifikat zu konfigurieren.

extensionCert

Objekt, das ein `contentSecret`-Objekt mit `name`- und `key`-Zeichenfolgen enthält. Das `contentSecret`-Objekt verweist auf ein geheimes Kubernetes-Objekt, das für ein TLS-Zertifikat erstellt wurde.

```
...
    variables:
    #extensionCert specifies the cert and key for Extensions Controller
    #self-signed issuer and certificates must be created in advance
    - name: extensionCert
      value:
        contentSecret:
          #name specifies the name of secret
          name: string
          #key specifies the content of tls\.crt in the secret's data map
          key: string
```

kubeAPIServerFQDNs

Verwenden Sie die Variable `kubeAPIServerFQDNs`, um einen Cluster mit einem FQDN zu konfigurieren.

kubeAPIServerFQDNs

Array aus einem oder mehreren vollqualifizierten Domänennamen (FQDN).

Das erzeugte Kubernetes-API-Zertifikat enthält alle FQDNs, die Sie in der Variable `kubeAPIServerFQDNs` angegeben haben. Das System befüllt kubeconfig mit dem ersten FQDN in der Liste und geht davon aus, dass dieser aufgelöst werden kann. Wenn Sie einen anderen FQDN in der Liste verwenden möchten, können Sie die erzeugte kubeconfig-Datei mit dem gewünschten FQDN aus der Variablenliste manuell bearbeiten.

Weitere Informationen dazu finden Sie unter v1beta1-Beispiel: Cluster mit FQDN.

nodePoolLabels

Verwenden Sie die `nodePoolLabels`-Variable, um Bezeichnungen für Worker-Knoten zu konfigurieren.

nodePoolLabels

Array aus einem oder mehreren Objekten, wobei jedes Objekt ein Schlüssel/Wert-Paar enthält, beides Zeichenfolgen.

Mithilfe von Bezeichnungen können Sie Systemobjekte gemäß Ihren Anforderungen organisieren, um die Abfrage und Berichterstellung zu vereinfachen. Informationen zur Nutzung finden Sie in der Dokumentation zu Kubernetes-Bezeichnungen.

nodePoolTaints

Verwenden Sie die `nodePoolTaints`-Variable, um Taints auf Worker-Knoten anzuwenden.

nodePoolTaints

Array von Objekten. Jedes Objekt enthält ein Merkmal (Taint), das für Worker-Knoten gilt.

Jedes Objekt enthält einen `key` (Zeichenfolge), einen `value` (Zeichenfolge) und eine `effect` (Zeichenfolge). Das Feld `timeAdded` wird beim Erstellen oder Aktualisieren durch das System aufgefüllt.

nodePoolVolumes

Verwenden Sie die `nodePoolVolumes`-Variable, um dauerhafte Volumes für Clusterknoten anzugeben.

nodePoolVolumes

Optionales Array von Objekten, von denen jedes `name`, `storageClass` und `mountPath` enthält, die jeweils Zeichenfolgen sind, und ein optionales `capacity`-Objekt, das eine `storage`-Zeichenfolge enthält.

```
...
    variables:
      #nodePoolVolumes is an optional set of PVCs to create and
      #attach to each node; use for high-churn components like containerd
      - name: nodePoolVolumes
        value: |
          #name of the PVC to be used as the suffix (node.name)
          - name: etcd
            #mountPath is the directory where the volume device is mounted
            #takes the form /dir/path
            mountPath: /var/lib/containerd
            #storageClass is the storage class to use for the PVC
            storageClass: tkgs-storage-profile
            #capacity is the PVC storage capacity
            capacity:
```

```
                #storage sets the capacity for the disk volume
                #if not specified defaults to storageClass capacity
                storage: 4Gi
```

ntp

Verwenden Sie die `ntp`-Variable, um einen NTP-Server für den Cluster zu konfigurieren.

ntp

Zeichenfolge, die den FQDN oder die IP-Adresse eines NTP-Servers darstellt.

Die NTP-Variable gibt den Domänennamen des NTP-Servers an, wie im Beispiel gezeigt. Der NTP-Server wird bei der Clustererstellung in die Clustervariable eingefügt. Nach der Clustererstellung verwalten Sie die Rotation des Servernamens oder ein beliebiges Update, indem Sie die Clustervariable manuell aktualisieren.

```
...
    variables:
    - name: ntp
      value: time1.vmware.com
```

podSecurityStandard

Verwenden Sie die Variable `podSecurityStandard`, um clusterweite Pod-Sicherheit zu konfigurieren.

Hinweis Hierbei handelt es sich um eine neue Variable für vSphere 8 Update 3 (TKG-Dienst 3.0).

podSecurityStandard

Mit TKr v1.26 und höher werden Pod-Sicherheitseinschränkungen (PSA) auf Namespace-Ebene standardmäßig mithilfe von Anmerkungsbezeichnungen erzwungen. Weitere Informationen hierzu finden Sie unter Konfigurieren von PSA für TKR 1.25 und höher.

Alternativ können Sie mithilfe der Variable `podSecurityStandard` clusterweite PSA konfigurieren, wenn Sie einen v1beta1-Cluster bereitstellen oder aktualisieren.

Die Variable `podSecurityStandard` kann folgendermaßen implementiert werden:

```
...
variables:
- name: podSecurityStandard
  value:
    deactivated: DEACTIVATED
    audit: AUDIT-PROFILE
    enforce: ENFORCE-PROFILE
    warn: WARN-PROFILE
    auditVersion: AUDIT-VERSION
    enforceVersion: ENFORCE-VERSION
    warnVersion: WARN-VERSION
    exemptions:
      namespaces: [EXEMPT-NS]
```

Dabei gilt:

- Der Wert DEACTIVATED ist auf `false` (Standardeinstellung) festgelegt, um clusterweite PSA anzuwenden, andernfalls lautet er auf `true`.
- Bei dem Wert *-PROFILE handelt es sich um das PSA-Profil für jeden Modus, der auf `"privileged"`, `"baseline"` oder `"restricted"` (Standardeinstellung) lauten kann.
- Der Wert *-VERSION stellt die Kubernetes-Version für jeden Modus dar, wie z. B. `"v1.26"`. Der Wert `"latest"` ist der Standardwert.
- Bei dem Wert EXEMPT-NS handelt es sich um eine kommagetrennte Liste von Namespaces, die aus der PSA-Steuerung ausgeschlossen werden sollen.

Hinweis System-Namespaces werden aus der Pod-Sicherheit ausgeschlossen, einschließlich kube-system, tkg-system und vmware-system-cloud-provider.

Wenn Sie die Variable `podSecurityStandard` nicht implementieren, wird das PSA-Standardverhalten beibehalten. Wenn Sie die Variable `podSecurityStandard` in die Clusterspezifikation aufnehmen, übernehmen die Variableneinstellungen die Steuerung, einschließlich der Standardwerte, es sei denn, sie werden überschrieben.

Im folgenden Beispiel werden die Standardwerte angezeigt.

```
...
    variables:
      - name: podSecurityStandard
        value:
          enforce: "restricted"
          enforce-version: "latest"
```

Das folgende Beispiel enthält Überwachungsprotokolle und Warnungen zur Erkennung von Arbeitslasten, die die Best Practices für die Pod-Härtung nicht einhalten, sondern lediglich eine minimal restriktive Richtlinie („Baseline“) erzwingen, die bekannte Berechtigungseskalationen verhindert.

```
...
    variables:
      - name: podSecurityStandard
        value:
          audit: "restricted"
          warn: "restricted"
          enforce: "baseline"
```

Im folgenden Beispiel wird eine eingeschränkte Richtlinie erzwungen. Ein bestimmter Namespace ist jedoch davon ausgenommen.

```
...
    variables:
      - name: podSecurityStandard
        value:
          audit: "restricted"
```

```
          warn: "restricted"
          enforce: "restricted"
          exemptions:
            namesaces: ["privileged-workload-ns"]
```

Im folgenden Beispiel beschränkt sich die Erzwingung auf eine bestimmte TKr-Version.

```
...
variables:
  - name: podSecurityStandard
    value:
      audit-version: "v1.26"
      warn-version: "v1.26"
      enforce-version: "v1.26"
```

Weitere Beispiele finden Sie unter Pod Security Standards in der Kubernetes-Dokumentation.

Proxy

Verwenden Sie die `proxy`-Variable, um einen Proxyserver für den Cluster zu konfigurieren.

Proxy

Objekt mit Parametern, die auf einen Proxyserver für ausgehende Clusterverbindungen verweisen.

Zu den erforderlichen `proxy`-Parametern gehören `httpProxy`, `httpsProxy` und `noProxy`. Alle drei Felder sind erforderlich, wenn Sie die Variable `proxy` in die Clusterdefinition aufnehmen.

Die Felder `httpProxy` und `httpsProxy` enthalten Zeichenfolgenwerte, die auf den URI eines Proxyservers verweisen, der für die Verwaltung von ausgehenden HTTP- und HTTPS-Verbindungen vom TKG-Cluster konfiguriert ist. Sie können mithilfe von HTTP eine Verbindung zum Proxy-Server herstellen. HTTPS-Verbindungen werden nicht unterstützt.

Das Feld `noProxy` ist ein Array von Zeichenfolgen. Rufen Sie die `noProxy`-Werte über das Supervisor-Arbeitslastnetzwerk ab. Im Feld `noProxy` müssen Sie das Namespace-Netzwerk sowie die Ingress- und Egress-Subnetze angeben.

Das Dienste-Subnetz müssen Sie nicht in das Feld `noProxy` eingeben. Der TKG-Cluster interagiert nicht mit diesem Subnetz.

Sie müssen `clusterNetwork.services.cidrBlocks` und `clusterNetwork.pods.cidrBlocks` nicht in das Feld `noProxy` aufnehmen. Diese Endpoints werden nicht automatisch mit einem Proxy verbunden.

Sie müssen localhost und `127.0.0.1` nicht in das Feld `noProxy` aufnehmen. Diese Endpoints werden nicht automatisch mit einem Proxy verbunden.

```
...
    variables:
    #proxy specifies a proxy server to be used for the cluster
    #if omitted no proxy is configured
    - name: proxy
      value:
```

```
        #httpProxy is the proxy URI for HTTP connections
        #to endpoints outside the cluster
        httpProxy: http://<user>:<pwd>@<ip>:<port>
        #httpsProxy is the proxy URL for HTTPS connections
        #to endpoints outside the cluster
        httpsProxy: http://<user>:<pwd>@<ip>:<port>
        #noProxy is the list of destination domain names, domains,
        #IP addresses, and other network CIDRs to exclude from proxying
        #must include Supervisor Pod, Egress, Ingress CIDRs
        noProxy: [array of strings, comma-separated]
```

storageClass

Verwenden Sie die `storageClass`-Variable, um eine Speicherklasse für den Cluster zu konfigurieren.

storageClass

Zeichenfolge, die dem Namen eines vSphere-Speicherprofils entspricht, das dem vSphere-Namespace zugewiesen wurde, in dem der TKG-Cluster bereitgestellt wird.

```
...
    variables:
    - name: storageClass
      value: tkgs-storage-profile
```

Verwenden Sie den folgenden Befehl zum Auflisten verfügbarer Speicherklassen:

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

Oder, wenn Sie über vSphere-Administratorrechte verfügen:

```
kubectl describe storageclasses
```

storageClasses

Verwenden Sie die `storageClasses`-Variable, um ein Array von Speicherklassen für den Cluster zu konfigurieren.

storageClasses

Array aus einer oder mehreren Zeichenfolgen, wobei jede Zeichenfolge der Name eines vSphere-Speicherprofils ist, das dem vSphere-Namespace zugewiesen wurde, in dem der TKG-Cluster bereitgestellt wird.

```
...
    variables:
    - name: storageClasses
      value: [tkg2-storage-profile, tkg2-storage-profile-latebinding]
```

Verwenden Sie den folgenden Befehl zum Auflisten verfügbarer Speicherklassen:

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

Oder, wenn Sie über vSphere-Administratorrechte verfügen:

```
kubectl describe storageclasses
```

TKR_DATA

Verwenden Sie die `TKR_DATA`-Variable, um TKR-Informationen anzugeben.

TKR_DATA

Objekt, das Sie zum Angeben der TKR-Version und anderer Details verwenden.

`version` ist eine Zeichenfolge im TKR NAME-Format, die von Clusterknoten verwendet wird.

Nur TKRs, die nicht über die `legacy-tkr`-Bezeichnung verfügen, sind mit TKG auf vSphere 9 Supervisor kompatibel. Weitere Informationen hierzu finden Sie unter Verwenden von Kubernetes-Versionen mit TKG-Dienstclustern.

Das Standardbetriebssystem ist PhotonOS. Verwenden Sie Anmerkungen, um die v1beta1-Beispiel: Cluster mit Ubuntu-TKR anzugeben.

trust

Verwenden Sie die `trust`-Variable, um ein oder mehrere vertrauenswürdige CA-Zertifikate für den Cluster anzugeben.

trust

Objekt zum Hinzufügen von TLS-Zertifikaten zum Cluster, entweder zusätzliche Zertifizierungsstellen oder Endzertifikate.

Der Wert lautet `additionalTrustedCAs` und enthält ein Array von Zeichenfolgen. Beispiel:

```
#trust-example
    variables:
      - name: trust
        value:
          additionalTrustedCAs:
          - name: additional-ca-1
          - name: additional-ca-2
          - name: additional-ca-N
```

Der Wert jeder Zeichenfolge ist der benutzerdefinierte Name für das Datenzuordnungsfeld im geheimen Kubernetes-Schlüssel, der das doppelt Base64-codierte CA-Zertifikat im PEM-Format enthält. Beispiel:

```
#secret-example
apiVersion: v1
data:
  additional-ca-1: TFMwdExTMUNSGlSzZ3Jaa...VVNVWkpRMEMwdExTMHRDZz09
```

```
kind: Secret
metadata:
  name: cluster01-user-trusted-ca-secret
  namespace: tkgs-cluster-ns
type: Opaque
```

Ein häufiger Anwendungsfall für die vertrauenswürdige Variable besteht in der Integration eines v1beta1-Clusters in eine private Containerregistrierung. Informationen dazu finden Sie unter Integrieren von TKG-Dienst-Clustern in eine private Containerregistrierung

Benutzer

Verwenden Sie die `user`-Variable, um die Anmeldedaten für Clusterbenutzer anzugeben.

Benutzer

Objekt, das ein passwordSecret-Objekt mit Namen und Schlüsselzeichenfolgen sowie eine sshAuthorizedKey-Zeichenfolge enthält. Sie können diese Variable verwenden, um den SSH-Schlüssel eines Benutzers zu Clusterknoten für den Remote-SSH-Zugriff hinzuzufügen.

Die Benutzervariable gibt die SSH-Anmeldedaten an, einschließlich Kennwort und autorisierter Schlüssel. Der Benutzername lautet standardmäßig `vmware-system-user`. Das Kennwort muss gehasht und in einem geheimen Schlüssel im selben Namespace gespeichert werden, in dem der Cluster bereitgestellt wird. Das Objekt `passwordSecret` verweist auf diesen geheimen Schlüssel. Unter Linux können Sie beispielsweise mit `mkpasswd --method=SHA-512 --rounds=4096` einen sicheren Hash generieren. Weitere Informationen finden Sie unter Einschließen von Benutzern und Gruppen.

```
...
    variables:
    #user specifies an authorized user and credentials
    - name: user
      value:
        #passwordSecret is an object that contains a Kubernetes secret and key
        passwordSecret:
          #name specifies the secret name
          name: string
          #key specifies the key value pair in the secret's data map
          key: string
        sshAuthorizedKey: string that is the base64-encoded public key
```

vmClass

Verwenden Sie die `vmClass`-Variable, um die VM-Klasse für Clusterknoten zu konfigurieren.

vmClass

Erforderliche Zeichenfolge, die dem Namen einer VM-Klasse zugeordnet ist, die an den vSphere-Namespace gebunden ist, in dem der TKG-Cluster bereitgestellt wird.

`vmClass` ist der Name der VirtualMachineClass, die die für Clusterknoten zu verwendenden virtuellen Hardwareeinstellungen beschreibt. Die VirtualMachineClass steuert die für den

Knoten verfügbare CPU und den Arbeitsspeicher sowie die Anforderungen und Grenzwerte für diese Ressourcen. Weitere Informationen finden Sie unter Verwenden von VM-Klassen mit TKG-Dienstclustern.

Sie können nur VM-Klassen verwenden, die mit dem vSphere-Namespace verknüpft sind, in dem der TKG-Cluster bereitgestellt wird. Verwenden Sie den Befehl `kubectl get virtualmachineclass`, um gebundene Klassen aufzulisten.

Sie können die Variable `vmClass` in unterschiedlichen Geltungsbereichen definieren, sodass Sie verschiedene VM-Klassen für Steuerungsebenenknoten und Knotenpool-Worker-Knoten verwenden können.

Beispiel: Hier überschreibt (`overrides`) eine Inline-`vmClass`-Variable die primäre Variable `vmClass` für diese spezifische machineDeployment-Topologie.

```
...
    workers:
      machineDeployments:
      - class: tkg-worker
        name: compute
        replicas: 3
        variables:
          overrides:
          - name: vmClass
            value: guaranteed-large
```

v1beta1-Beispiel: Standardcluster

In diesem Beispiel finden Sie Informationen zur Bereitstellung eines v1beta1-Clusters mit Standardeinstellungen.

v1beta1-Beispiel: Standardcluster

In der folgenden Beispiel-YAML wird ein Standardcluster basierend auf der Standard-ClusterClass über die v1beta1-API erstellt.

Dieses Beispiel stellt die Mindestkonfiguration dar, die zum Erstellen eines Clusters mithilfe der v1beta1-API erforderlich ist. Das Beispiel enthält Beschreibungen für jedes Feld. Weitere Informationen finden Sie im Quellcode.

Beachten Sie die folgenden Informationen zu diesem Beispiel:

- Im Gegensatz zur TanzuKubernetesCluster v1alpha3-API müssen Sie für die v1beta1-API das `clusterNetwork` angeben. Es gibt keine Standardnetzwerkeinstellung für den Clustertyp.
- Die standardmäßige ClusterClass ist `tanzukubernetescluster`. Diese ist hier dokumentiert: Cluster-API v1beta1 .
- Sie können den Cluster mithilfe von `variables` anpassen, wobei es sich jeweils um ein Name/Wert-Paar handelt. Wie im Beispiel gezeigt, müssen Sie zumindest die VM und die Speicherklassen als Variablen angeben.

- Das Beispiel enthält auch eine technisch optionale Variable `defaultStorageClass`, da der Cluster bei vielen Arbeitslasten, einschließlich Tanzu-Paketen und Helm-Diagrammen, mit einer Standardspeicherklasse bereitgestellt werden muss.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
#define the cluster
metadata:
  #user-defined name of the cluster; string
  name: cluster-default
  #kubernetes namespace for the cluster; string
  namespace: tkg-cluster-ns
#define the desired state of cluster
spec:
  #specify the cluster network; required, there is no default
  clusterNetwork:
    #network ranges from which service VIPs are allocated
    services:
      #ranges of network addresses; string array
      #CAUTION: must not overlap with Supervisor
      cidrBlocks: ["198.51.100.0/12"]
    #network ranges from which Pod networks are allocated
    pods:
      #ranges of network addresses; string array
      #CAUTION: must not overlap with Supervisor
      cidrBlocks: ["192.0.2.0/16"]
    #domain name for services; string
    serviceDomain: "cluster.local"
  #specify the topology for the cluster
  topology:
    #name of the ClusterClass object to derive the topology
    class: tanzukubernetescluster
    #kubernetes version of the cluster; format is TKR NAME
    version: v1.26.13---vmware.1-fips.1-tkg.3
    #describe the cluster control plane
    controlPlane:
      #number of control plane nodes
      #integer value 1 or 3
      #NOTE: Production clusters require 3 control plane nodes
      replicas: 3
    #describe the cluster worker nodes
    workers:
      #specifies parameters for a set of worker nodes in the topology
      machineDeployments:
        #node pool class used to create the set of worker nodes
        - class: node-pool
          #user-defined name of the node pool; string
          name: node-pool-1
          #number of worker nodes in this pool; integer 0 or more
          replicas: 3
    #customize the cluster
    variables:
      #virtual machine class type and size for cluster nodes
      - name: vmClass
```

```
        value: guaranteed-medium
      #persistent storage class for cluster nodes
      - name: storageClass
        value: tkg-storage-policy
      # default storageclass for control plane and worker node pools
      - name: defaultStorageClass
        value: tkg-storage-policy
```

v1beta1-Beispiel: Benutzerdefinierter Cluster basierend auf der standardmäßigen ClusterClass

In diesem Beispiel finden Sie Informationen zur Bereitstellung eines v1beta1-Clusters mit benutzerdefinierten Einstellungen.

v1beta1-Beispiel: Benutzerdefinierter Cluster basierend auf der standardmäßigen ClusterClass

Die folgende Beispiel-YAML veranschaulicht, wie Sie mit der v1beta1-API einen Cluster mit verschiedenen benutzerdefinierten Einstellungen über Variablen bereitstellen. Dieses Beispiel basiert auf dem v1beta1-Beispiel: Standardcluster.

In diesem Beispiel werden Variablen für dauerhafte Volumes auf Worker-Knoten für Komponenten mit hohem Durchsatz wie containerd und kubelet verwendet. Darüber hinaus wird die Variable `vmClass` zweimal deklariert. Die in `workers.machineDeployments` deklarierte Variable `vmClass` überschreibt die global deklarierte Variable `vmClass`, sodass Worker-Knoten mit einer größeren VM-Klasse bereitgestellt werden.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster-custom
  namespace: tkg-cluster-ns
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.51.100.0/12"]
    pods:
      cidrBlocks: ["192.2.0.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.25.7---vmware.3-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          name: node-pool-1
          replicas: 3
          variables:
            overrides:
            - name: vmClass
```

```
            value: guaranteed-xlarge
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkg-storage-profile
      - name: defaultStorageClass
        value: tkg-storage-profile
      - name: nodePoolVolumes
        value:
          - name: containerd
            capacity:
              storage: 50Gi
            mountPath: /var/lib/containerd
            storageClass: tkg-storage-profile
          - name: kubelet
            capacity:
              storage: 50Gi
            mountPath: /var/lib/kubelet
            storageClass: tkg-storage-profile
```

v1beta1-Beispiel: Cluster mit Calico-CNI

In diesem Beispiel finden Sie Informationen zur Bereitstellung eines v1beta1-Clusters mit der Calico-CNI anstelle der standardmäßigen Antrea-CNI. Darüber hinaus enthält dieses Beispiel Informationen zur Anpassung eines oder mehrerer TKR-Pakete für einen Cluster.

v1beta1-Beispiel: Cluster mit benutzerdefinierter CNI

Die folgende Beispiel-YAML veranschaulicht, wie Sie mit der v1beta1-API einen Cluster mit einer benutzerdefinierten CNI bereitstellen. Dieses Beispiel basiert auf dem v1beta1-Beispiel: Standardcluster.

Wie in der ClusterClass von `tanzukubernetescluster` definiert ist, lautet die Standard-CNI Antrea. Die andere unterstützte CNI ist Calico. Um die CNI von Antrea in Calico zu ändern, überlasten Sie die Standard-CNI, indem Sie eine benutzerdefinierte Ressource `ClusterBootstrap` erstellen, die auf eine benutzerdefinierte Ressource `CalicoConfig` verweist.

Die benutzerdefinierte Ressource `ClusterBootstrap` enthält den Block `spec.cni.refName` mit dem von der TKR stammenden Wert. (Weitere Informationen zum Abrufen des Werts für dieses Feld finden Sie unter TKr-Pakete.) Der Wert `ClusterBootstrap` überschreibt den Standardwert in der ClusterClass und wird bei der Erstellung des Clusters von der Cluster-API (CAPI) übernommen. Der Name der benutzerdefinierten Ressource `ClusterBootstrap` muss mit dem `Cluster` identisch sein.

Hinweis Das Beispiel wird als einzelne YAML-Datei bereitgestellt und kann aber in einzelne Dateien aufgeteilt werden. Wenn Sie dies tun, müssen Sie sie in der folgenden Reihenfolge erstellen: zuerst die benutzerdefinierte Ressource `CalicoConfig`, dann die benutzerdefinierte Ressource `ClusterBootstrap` und danach den Cluster `cluster-calico`.

```
---
apiVersion: cni.tanzu.vmware.com/v1alpha1
kind: CalicoConfig
metadata:
  name: cluster-calico
spec:
  calico:
    config:
      vethMTU: 0
---
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: ClusterBootstrap
metadata:
  annotations:
    tkg.tanzu.vmware.com/add-missing-fields-from-tkr: v1.23.8---vmware.2-tkg.2-zshippable
  name: cluster-calico
spec:
  cni:
    refName: calico.tanzu.vmware.com.3.22.1+vmware.1-tkg.2-zshippable
    valuesFrom:
      providerRef:
        apiGroup: cni.tanzu.vmware.com
        kind: CalicoConfig
        name: cluster-calico
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster-calico
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.51.100.0/12"]
    pods:
      cidrBlocks: ["192.0.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.23.8---vmware.2-tkg.2-zshippable
    controlPlane:
```

```
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          name: node-pool-1
          replicas: 3
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkg2-storage-policy
```

v1beta1-Beispiel: Cluster mit Ubuntu-TKR

In diesem Beispiel finden Sie Informationen zur Bereitstellung eines v1beta1-Clusters, der die Ubuntu-Edition einer Tanzu Kubernetes-Version verwendet.

v1beta1-Beispiel: Cluster mit Ubuntu-TKR

Die folgende Beispiel-YAML veranschaulicht, wie Sie mit der v1beta1-API einen Cluster bereitstellen, der die Ubuntu-Edition der angegebenen TKR verwendet. Dieses Beispiel basiert auf dem v1beta1-Beispiel: Standardcluster.

Standardmäßig wird Photon OS für Clusterknoten verwendet. Wenn die TKR-Version mehrere OSImages unterstützt, fügen Sie die Anmerkung `run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu` in die Clusterspezifikation ein, um Ubuntu anstelle von Photon zu verwenden. Wenn Sie die Ubuntu-Edition der TKr verwenden, müssen Sie die vollständige VERSION-Zeichenfolge angeben und die Betriebssystemanmerkung in die Clusterspezifikation aufnehmen. Weitere Informationen zu TKRs finden Sie unter Kapitel 5 Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster-ubuntu
  namespace: tkg-cluster-ns
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.51.100.0/12"]
    pods:
      cidrBlocks: ["192.0.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.25.7---vmware.3-fips.1-tkg.1
    controlPlane:
      replicas: 3
      metadata:
        annotations:
          run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
    workers:
```

```
      machineDeployments:
        - class: node-pool
          name: node-pool-1
          replicas: 3
          metadata:
            annotations:
              run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
        - class: node-pool
          name: node-pool-2
          replicas: 3
          metadata:
            annotations:
              run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
        - class: node-pool
          name: node-pool-3
          replicas: 3
          metadata:
            annotations:
              run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkg-storage-policy
```

v1beta1-Beispiel: Cluster mit FQDN

In diesem Beispiel finden Sie Informationen zur Bereitstellung eines TKGS-Clusters mit einem oder mehreren vollqualifizierten Domänennamen (FQDN).

Unterstützung für FQDN

Sie können die v1beta1-API verwenden, um einen TKG-Cluster mit einem vollqualifizierten Domänennamen (FQDN) bereitzustellen. Die v1beta1-API des Clusters enthält eine Variable mit dem Namen `kubeAPIServerFQDNs` und mindestens einer FQDN-Zeichenfolge, die im TLS-Zertifikat für den Kubernetes-API-Server erzeugt werden muss.

Wenn der Befehl `kubectl vsphere login` für einen Cluster mit konfiguriertem FQDN ausgegeben wird, wählt der Authentifizierungsdienst den ersten FQDN-Eintrag in der Liste aus und fügt ihn der kubeconfig-Datei als bevorzugte Option für die Interaktion mit dem Cluster hinzu. Es wird davon ausgegangen, dass der erste FQDN in der Liste aufgelöst werden kann. Änderungen an der Clusteranmeldung sind nicht erforderlich.

Das erzeugte Kubernetes-API-Zertifikat enthält alle FQDNs, die Sie in der Variable `kubeAPIServerFQDNs` angegeben haben. Das System verwendet nur den ersten FQDN in der Liste. Das System versucht nicht, den FQDN aufzulösen. Wenn Sie einen anderen FQDN in der Liste verwenden möchten, können Sie die erzeugte kubeconfig-Datei manuell bearbeiten und den gewünschten FQDN hinzufügen.

Anforderungen an den FQDN

Die Verwendung eines FQDN ist optional. Wenn Sie keinen FQDN verwenden, hat dies keine Auswirkungen auf die Funktionen. Die hier beschriebenen Funktionen sind für TKG-Arbeitslastcluster spezifisch. Informationen zur Verwendung eines FQDN mit Supervisor finden Sie in diesem Thema in der Dokumentation zu Supervisor.

Beachten Sie die folgenden Anforderungen für die Bereitstellung eines TKG-Clusters mit einem FQDN.

- Umgebungen mit vSphere 8.0 U2 P03 und höher
- Supervisor wurde auf die aktuelle Patch-Version aktualisiert
- Nur v1beta1-API-Cluster werden unterstützt. v1alpha3-API-Cluster werden nicht unterstützt
- DNS ist zur Auflösung des ausgewählten FQDN in eine gültige IP-Adresse konfiguriert

Wichtig Die FQDN-Funktion ist nur verfügbar, wenn Sie die v1beta1-API zur Bereitstellung eines CAPI-Clusters verwenden. Sie können keine TKC unter Verwendung der v1alpha3-API mit einem FQDN bereitstellen.

FQDN-Beispiel

Verwenden Sie die v1beta1-API des Clusters, um einen Cluster mit einem FQDN zu erstellen.

Das Feld `spec.topology.variables.kupeAPIServerFQDNs` ist ein Array von FQDNs.

Das System wählt den ersten FQDN in der Liste aus, der in diesem Beispiel auf `demo.fqdn.com` lautet.

```
#cluster-example-fqdn.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: tkg-cluster-fqdn
  namespace: tkg-ns
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.52.100.0/12"]
    pods:
      cidrBlocks: ["192.101.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.26.5+vmware.2-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          name: node-pool-01
          replicas: 3
    variables:
```

```
    - name: vmClass
      value: guaranteed-medium
    - name: storageClass
      value: tkgs-storage-class
    - name: defaultStorageClass
      value: tkg-storage-class
    - name: kubeAPIServerFQDNs
      value:
        - demo.fqdn.com
        - explore.fqdn.com
```

FQDN-Überprüfung

Führen Sie das folgende Verfahren aus, um sicherzustellen, dass der erste FQDN in der Variablenliste in der Datei `kubeconfig` enthalten ist und dass sich alle FQDNs in der Variablenliste im TLS-Zertifikat für den Kubernetes-API-Server befinden.

1 Melden Sie sich über Kubectl beim TKG-Cluster an.

```
kubectl vsphere login --server=SVCP IP or FQDN --vsphere-username USERNAME --tanzu-
kubernetes-cluster-name CLUSTER-NAME --tanzu-kubernetes-cluster-namespace VSPHERE-NS
```

2 So zeigen Sie den FQDN in der Datei `kubeconfig` an.

```
cat ~/.kube/config
```

3 Stellen Sie sicher, dass die erste FQDN-Variable in der Liste in der Datei `kubeconfig` enthalten ist.

Beispiel:

```
apiVersion: v1
clusters:
- cluster:
    insecure-skip-tls-verify: false
    server: https://10.199.155.77:6443
  name: 10.199.155.77
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJ...DQWRLZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXc
    ...
    CkdiL1pua09rOVVjT3BwSStCTE9ZZDR0RGd2eHo...QUp0SUUKLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    server: https://demo.fqdn.com:6443
  name: demo.fqdn.com
```

4 Rufen Sie mithilfe des vSphere Client die IP-Adresse für den TKGS-Cluster unter **Arbeitslastverwaltung > *Namespace* > Berechnen > Tanzu Kubernetes-Cluster > Adresse der Steuerungsebene** ab.

5 Erstellen Sie einen manuellen DNS-Eintrag in der lokalen `/etc/hosts`-Datei mit der IP-Adresse und dem FQDN.

Beispiel:

```
sudo vi /etc/hosts
127.0.0.1 localhost
127.0.1.1 ubuntu-client-vm
10.199.155.77 demo.fqdn.com
...
```

6 Verwenden Sie den Befehl `openssl s_client`, um das TLS-Zertifikat anzuzeigen.

```
echo | openssl s_client -servername hostname -connect FQDN:PORT 2>/dev/null | openssl x509
-text
```

Hierbei ist `FQDN` der erste FQDN in der Variablenliste `kubeAPIServerFQDNs`.

Beispiel:

```
echo | openssl s_client -servername hostname -connect demo.fqdn.com:6443 2>/dev/null |
openssl x509 -text
```

7 Im Feld `Subject Alternative Name` sind alle FQDNs enthalten.

```
X509v3 Subject Alternative Name:
    DNS:demo.fqdn.com, DNS:explore.fqdn.com, DNS:kubernetes, DNS:kubernetes.default,
DNS:kubernetes.default.svc, DNS:kubernetes.def
```

Da das TLS-Zertifikat für den Kubernetes-API-Server alle FQDNs in der Liste `kubeAPIServerFQDNs` enthält, können Sie die Datei `kubeconfig` manuell aktualisieren, um den zweiten (oder dritten usw.) FQDN in der Liste zu verwenden. Diese Vorgehensweise funktioniert unter der Voraussetzung, dass der FQDN aufgelöst werden kann.

v1beta1-Beispiel: Cluster in verschiedenen vSphere-Zonen

In diesem Beispiel finden Sie Informationen zur Bereitstellung eines v1beta1-Clusters auf dem in drei verschiedenen vSphere-Zonen bereitgestellten Supervisor.

v1beta1-Beispiel: Cluster in verschiedenen vSphere-Zonen

In der folgenden Beispiel-YAML wird ein Cluster über die v1beta1-API in einer vSphere-Zonentopologie bereitgestellt. Dieses Beispiel basiert auf dem v1beta1-Beispiel: Standardcluster.

In diesem Beispiel werden mehrere Worker-Knotenpools implementiert. Jeder Knotenpool verweist auf eine Fehlerdomäne, die einer vSphere-Zone zugeordnet ist. Weitere Informationen zu vSphere-Zonen finden Sie unter *Installieren und Konfigurieren der vSphere IaaS-Steuerungsebene*.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster-zoned
  namespace: tkg-cluster-ns
spec:
```

```
  clusterNetwork:
    services:
      cidrBlocks: ["198.51.100.0/12"]
    pods:
      cidrBlocks: ["192.0.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.25.7---vmware.3-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      #muliple node pools are used
      machineDeployments:
        - class: node-pool
          name: node-pool-1
          replicas: 3
          #failure domain the machines will be created in
          #maps to a vSphere Zone; name must match exactly
          failureDomain: vsphere-zone1
        - class: node-pool
          name: node-pool-2
          replicas: 3
          #failure domain the machines will be created in
          #maps to a vSphere Zone; name must match exactly
          failureDomain: vsphere-zone2
        - class: node-pool
          name: node-pool-3
          replicas: 3
          #failure domain the machines will be created in
          #maps to a vSphere Zone; name must match exactly
          failureDomain: vsphere-zone3
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkg-storage-policy
```

v1beta1-Beispiel: Cluster mit routingfähigem Pods-Netzwerk

Sie können die v1beta1-API verwenden, um ein Cluster-Netzwerk mit routingfähigen Pods zu erstellen. Dazu überschreiben Sie den Standardcluster mit benutzerdefinierten Konfigurationen für `AntreaConfig` und `VSphereCPIConfig`.

Informationen zu routingfähigen Pods-Netzwerken mithilfe der v1beta1-API

Die folgende Beispiel-YAML veranschaulicht, wie Sie mit der v1beta1-API einen Cluster mit der aktivierten Antrea RoutablePod-Funktion bereitstellen. Dieses Beispiel basiert auf dem v1beta1-Beispiel: Standardcluster.

Um die Routing-Pod-Funktion zu aktivieren, benötigt der Cluster `AntreaConfig` und `VSphereCPIConfig` mit einer speziellen Konfiguration.

Der `AntreaConfig` muss `trafficEncapMode: noEncap` und `noSNAT: true` festlegen.

`VSphereCPIConfig` muss `antreaNSXPodRoutingEnabled: true`, `mode: vsphereParavirtualCPI` und festlegen.

```
tlsCipherSuites:
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_
WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1
305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
```

Der Name der `AntreaConfig` muss genau als `<cluster-name>-antrea-package` formatiert werden. Der Name der `VSphereCPIConfig` muss genau als `<cluster-name>-vsphere-cpi-package` formatiert werden.

Erstellen Sie die Konfigurationsdateien, erstellen Sie das Clusterspezifikationsobjekt, das auf die Konfigurationsdateien verweist. Während der Clustererstellung werden die Konfigurationsdateien verwendet und um den Cluster bereitzustellen und die Standardkonfiguration zu überschreiben.

Erstellen eines routingfähigen Pod-Netzwerks: Supervisor-Konfiguration

Zum Erstellen eines routingfähigen Pod-Netzwerks ist eine Konfiguration auf dem Supervisor und im TKG-Cluster erforderlich.

Hinweis Supervisor muss mit NSX konfiguriert werden, um routingfähige Pods-Netzwerke zu verwenden. Sie können keine routingfähigen Pods mit dem VDS-Netzwerk verwenden.

So konfigurieren Sie ein routingfähiges Pod-Netzwerk auf Supervisor:

1 Erstellen Sie einen neuen vSphere-Namespace.

   Weitere Informationen finden Sie unter Erstellen eines vSphere-Namespace für das Hosting von TKG-Dienst-Clustern.

2 Aktivieren Sie das Kontrollkästchen **Supervisor-Netzwerkeinstellungen außer Kraft setzen**.

   Weitere Informationen finden Sie unter Überschreiben der Einstellungen für das Arbeitslastennetzwerk für einen vSphere-Namespace.

3 Konfigurieren Sie das routingfähige Pod-Netzwerk wie folgt.

Bereich	Beschreibung
NAT-Modus	Deaktivieren Sie diese Option, um die Netzwerkadressübersetzung (Network Address Translation, NAT) zu deaktivieren.
Namespace-Netzwerk	Befüllen Sie dieses Feld mit einem routingfähigen IP-Subnetz im Format „IP-Adresse/Bit“ (z. B. 10.0.0.6/16). NCP erstellt einen oder mehrere IP-Pools aus den für das Netzwerk angegebenen IP-Blöcken. Sie sollten mindestens eine /23-Subnetzgröße angeben. Wenn Sie beispielsweise ein routingfähiges /23-Subnetz mit einem /28-Subnetzpräfix angeben, erhalten Sie 32 Subnetze, die für einen Cluster mit 6 Knoten ausreichen sollten. Ein /24-Subnetz mit einem /28-Präfix ergibt nur 2 Subnetze, die nicht ausreichen. **Achtung** Stellen Sie sicher, dass sich das von Ihnen hinzugefügte routingfähige IP-Netzwerk nicht mit dem **CIDR der Dienste** überschneidet, der die IP-Adressen für Clusterknoten zuweist. Sie können den **CIDR der Dienste** unter **Supervisor > Konfigurieren > Netzwerk > Arbeitslastnetzwerk** überprüfen.
Namespace-Subnetzpräfix	Geben Sie beispielsweise ein Subnetzpräfix im Format „/28“ an. Das Subnetzpräfix wird verwendet, um das Pod-Subnetz für jeden Knoten aus dem Namespace-Netzwerk zu entfernen.

4 Klicken Sie auf **Erstellen**, um ein Netzwerk routingfähiger Pods zu erstellen.

Erstellen eines routingfähigen Pod-Netzwerks: TKG-Clusterkonfiguration

Die folgende Beispiel-YAML zeigt, wie ein v1beta1-Cluster mit einem Netzwerk routingfähiger Pods konfiguriert wird.

Wenn Sie im folgenden Beispiel gezeigt haben, müssen Sie den Abschnitt `spec.clusterNetwork.pod` aus der Clusterspezifikation entfernen, da die Pod-IP-Adressen von cloud-provider-vsphere zugeteilt werden.

Hinweis Das Beispiel wird als einzelne YAML-Datei bereitgestellt und kann aber in einzelne Dateien aufgeteilt werden. Wenn Sie dies tun, müssen Sie sie in der folgenden Reihenfolge erstellen: zuerst die benutzerdefinierte Ressource `AntreaConfig`, dann die benutzerdefinierte Ressource `VSphereCPIConfig` und danach den Cluster target-cluster.

```
---
apiVersion: cni.tanzu.vmware.com/v1alpha1
kind: AntreaConfig
metadata:
 name: target-cluster-antrea-package
spec:
```

```
 antrea:
   config:
     defaultMTU: ""
     disableUdpTunnelOffload: false
     featureGates:
       AntreaPolicy: true
       AntreaProxy: true
       AntreaTraceflow: true
       Egress: false
       EndpointSlice: true
       FlowExporter: false
       NetworkPolicyStats: false
       NodePortLocal: false
     noSNAT: true
     tlsCipherSuites:
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WI
TH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384
     trafficEncapMode: noEncap
---
apiVersion: cpi.tanzu.vmware.com/v1alpha1
kind: VSphereCPIConfig
metadata:
 name: target-cluster-vsphere-cpi-package
spec:
 vsphereCPI:
   antreaNSXPodRoutingEnabled: true
   insecure: false
   mode: vsphereParavirtualCPI
   tlsCipherSuites:
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_
WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1
305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
 name: target-cluster
spec:
 clusterNetwork:
   services:
     cidrBlocks: ["198.51.100.0/12"]
   serviceDomain: "cluster.local"
 topology:
   class: tanzukubernetescluster
   version: v1.25.7---vmware.3-fips.1-tkg.1
   controlPlane:
     replicas: 3
   workers:
     machineDeployments:
       - class: node-pool
         name: node-pool-1
         replicas: 3
   variables:
```

```
    - name: vmClass
      value: guaranteed-medium
    - name: storageClass
      value: tkg2-storage-policy
```

v1beta1-Beispiel: Cluster mit zusätzlichen vertrauenswürdigen CA-Zertifikaten für SSL/TLS

In diesem Beispiel finden Sie Informationen zur Bereitstellung eines v1beta1-Clusters mit mindestens einem zusätzlichen vertrauenswürdigen CA-Zertifikat.

v1beta1-Beispiel: Cluster mit zusätzlichen vertrauenswürdigen CA-Zertifikaten

Die Cluster-API v1beta1 stellt die trust-Variable für die Bereitstellung eines Clusters mit mindestens einem zusätzlichen vertrauenswürdigen CA-Zertifikat bereit.

Tabelle 7-1. Vertrauenswürdige v1beta1-API-Variable

Bereich	Beschreibung
`trust`	Abschnittsmarkierung. Akzeptiert keine Daten.
`additionalTrustedCAs`	Abschnittsmarkierung. Enthält ein Array von Zertifikaten mit `name` für jedes Zertifikat.
`name`	Der benutzerdefinierte Name für das Zuordnungsfeld `data` im geheimen Kubernetes-Schlüssel, der das doppelt Base64-codierte CA-Zertifikat im PEM-Format enthält. **Hinweis** Doppelte Base64-Codierung ist erforderlich. Wenn der Inhalt nicht doppelt Base64-codiert ist, kann die resultierende PEM-Datei nicht verarbeitet werden.

In folgendem Beispiel wird das Hinzufügen des geheimen Kubernetes-Schlüssels mit einem CA-Zertifikat zu einer v1beta1-API-Clusterspezifikation veranschaulicht.

```
#cluster-with-trusted-private-reg-cert.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster01
  namespace: tkgs-cluster-ns
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.52.100.0/12"]
    pods:
      cidrBlocks: ["192.101.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
    class: tanzukubernetescluster
    version: v1.26.5+vmware.2-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
```

```
      machineDeployments:
        - class: node-pool
          name: node-pool-01
          replicas: 3
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkgs-storage-profile
      - name: defaultStorageClass
        value: tkgs-storage-profile
      - name: trust
        value:
          additionalTrustedCAs:
          - name: additional-ca-1
```

In folgendem Beispiel wird der geheime Kubernetes-Schlüssel erläutert, der ein zusätzliches vertrauenswürdiges CA-Zertifikat enthält.

```
#additional-ca-1.yaml
apiVersion: v1
data:
  additional-ca-1: TFMwdExTMUNSG1SzZ3Jaa...VVNVWkpRMEMwdExTMHRDZz09
kind: Secret
metadata:
  name: cluster01-user-trusted-ca-secret
  namespace: tkgs-cluster-ns
type: Opaque
```

Dabei gilt:

- Der Wert der `data`-Zuordnung des geheimen Schlüssels stellt eine benutzerdefinierte Zeichenfolge dar, die als Name des CA-Zertifikats fungiert (in diesem Beispiel `additional-ca-1`), dessen Wert das doppelt Base64-codierte CA-Zertifikat im PEM-Format ist.
- Versehen Sie den geheimen Schlüssel im Abschnitt `metadata` mit dem Namen *`CLUSTER-NAME`*`-user-trusted-ca-secret`, wobei *CLUSTER-NAME* als Name des Clusters fungiert. Dieser geheime Schlüssel muss im selben vSphere-Namespace wie der Cluster erstellt werden.

So versehen Sie den Inhalt des CA-Zertifikats mit einer doppelten base64-Codierung.

- Linux: `base64 -w 0 ca.crt | base64 -w 0`
- Windows: https://www.base64encode.org/.

Verfahren: Neuer Cluster

Führen Sie folgendes Verfahren aus, um mindestens ein zusätzliches vertrauenswürdiges CA-Zertifikat in einen neuen TKGS-Cluster aufzunehmen.

1 Verwenden Sie für den Inhalt eines CA-Zertifikats eine doppelte Base64-Codierung.

2 Erstellen Sie einen geheimen Kubernetes-Schlüssel mit dem Datenzuordnungsnamen, dessen Wert ein doppelt Base64-codiertes CA-Zertifikat im PEM-Format darstellt.

3 Befüllen Sie in der Clusterspezifikation die Variable `trust.additionalTrustedCAs` mit dem Namen der Datenzuordnung.

4 Stellen Sie den Cluster wie gewohnt bereit.

   Weitere Informationen hierzu finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl.

5 Nach der erfolgreichen Bereitstellung des Clusters wird das von Ihnen hinzugefügte CA-Zertifikat vom Cluster als vertrauenswürdig eingestuft.

Verfahren: Vorhandener Cluster

Führen Sie das folgende Verfahren aus, um einem vorhandenen Cluster mindestens ein zusätzliches vertrauenswürdiges CA-Zertifikat hinzuzufügen.

1 Verwenden Sie für den Inhalt eines CA-Zertifikats eine doppelte Base64-Codierung.

2 Erstellen Sie einen geheimen Kubernetes-Schlüssel mit dem Datenzuordnungsnamen, dessen Wert ein doppelt Base64-codiertes CA-Zertifikat im PEM-Format darstellt.

3 Stellen Sie sicher, dass Sie die kubectl-Bearbeitung konfiguriert haben.

   Weitere Informationen hierzu finden Sie unter Konfigurieren eines Texteditors für Kubectl.

4 Bearbeiten Sie die Clusterspezifikation.

```
kubectl edit tanzukubernetescluster/tkgs-cluster-name
```

5 Fügen Sie den Abschnitt `trust.additionalTrustedCAs` zur Spezifikation hinzu.

6 Befüllen Sie das Feld `additionalTrustedCAs` mit dem Namen der Datenzuordnung im geheimen Schlüssel, der das doppelt Base64-codierte CA-Zertifikat im PEM-Format enthält.

7 Speichern Sie die Änderungen im Texteditor und stellen Sie sicher, dass die Änderungen von kubectl registriert wurden.

```
kubectl edit cluster/tkgs-cluster-name
cluster.run.tanzu.vmware.com/tkgs-cluster-name edited
```

8 Wenn ein paralleles Update für den Cluster initiiert wird, werden die zusätzlichen vertrauenswürdigen CA-Zertifikate hinzugefügt.

   Weitere Informationen hierzu finden Sie unter Grundlegendes zum Modell für parallele Updates für TKG-Dienstcluster.

Überprüfen zusätzlicher vertrauenswürdiger CA-Zertifikate

Die zusätzlichen vertrauenswürdigen CA-Zertifikate, die dem Cluster hinzugefügt wurden, sind in der Kubeconfig-Datei für den Cluster enthalten.

Rotieren des Zertifikats

Erstellen Sie zum Rotieren eines Zertifikats einen neuen geheimen Schlüssel und bearbeiten Sie die Clusterspezifikation mit dem entsprechenden Wert. Auf diese Weise wird ein paralleles Update des Clusters ausgelöst.

Hinweis Änderungen am `CLUSTER-NAME-user-trusted-ca-secret` werden vom System nicht überwacht. Änderungen am Wert der `data`-Zuordnung spiegeln sich nicht im Cluster wider. Sie müssen einen neuen geheimen Schlüssel erstellen, dessen Angaben eine Zuordnung von `name` zu `trust.additionalTrustCAs` beschreiben.

Fehlerbehebung bei zusätzlichen vertrauenswürdigen CA-Zertifikaten

Weitere Informationen hierzu finden Sie unter Fehlerbehebung bei zusätzlichen vertrauenswürdigen Zertifizierungsstellen.

Anwendungsbeispiele

Der häufigste Anwendungsfall besteht im Hinzufügen einer zusätzlichen vertrauenswürdigen Zertifizierungsstelle zum Herstellen einer Verbindung mit einer Containerregistrierung. Weitere Informationen hierzu finden Sie unter Integrieren von TKG-Dienst-Clustern in eine private Containerregistrierung.

v1beta1-Beispiel: Cluster basierend auf einer benutzerdefinierten ClusterClass (vSphere 8 U2- und höherer Workflow)

Lesen Sie diese Anweisungen, um einen TKG-Cluster basierend auf einer benutzerdefinierten ClusterClass bereitzustellen. Beachten Sie, dass diese Anweisungen spezifisch für vSphere 8 U2- und höhere Umgebungen gelten.

Voraussetzungen

Das Verfahren für die Bereitstellung eines TKG-Clusters basierend auf einer benutzerdefinierten ClusterClass wurde für die Version vSphere 8 U2 aktualisiert.

Beachten Sie die folgenden Voraussetzungen.

- vSphere 8 U2- und höhere Umgebung
- Arbeitslastverwaltung aktiviert
- Supervisor konfiguriert

- Ubuntu-Client mit installierten Kubernetes-CLI-Tools für vSphere

Achtung Die benutzerdefinierte ClusterClass ist eine experimentelle Kubernetes-Funktion gemäß der Upstream-Cluster-API-Dokumentation. Aufgrund der im Zusammenhang mit der benutzerdefinierten ClusterClass verfügbaren Bandbreite an Anpassungen kann VMware nicht alle möglichen Anpassungen testen oder validieren. Die Kunden sind für das Testen, Validieren und Beheben von Fehlern ihrer benutzerdefinierten ClusterClass-Cluster verantwortlich. Sie können Support-Tickets für ihre benutzerdefinierten ClusterClass-Cluster öffnen. VMware Support beschränkt sich jedoch auf eine Unterstützung nach bestem Wissen und kann nicht garantieren, dass alle Probleme behoben werden, die bei benutzerdefinierten ClusterClass-Clustern auftreten. Die Kunden sollten sich diese Risiken bewusst machen, bevor sie benutzerdefinierte ClusterClass-Cluster in Produktionsumgebungen bereitstellen.

Workflows auf hoher Ebene

Die Workflows auf hoher Ebene sehen wie folgt aus.

Der folgende Workflow ist alles, was Sie für die ersten Schritte benötigen.

Schritt	Aufgabe	Anleitung
1	Erstellen Sie eine benutzerdefinierte ClusterClass, indem Sie die standardmäßige ClusterClass klonen.	1: Erstellen einer benutzerdefinierten ClusterClass
2	Stellen Sie einen neuen TKG-Cluster basierend auf der benutzerdefinierten ClusterClass bereit und vergewissern Sie sich, dass alle Clusterknoten ordnungsgemäß hochgefahren werden.	2: Erstellen eines TKG-Clusters basierend auf der benutzerdefinierten ClusterClass

Im folgenden Workflow können Sie Änderungen an der benutzerdefinierten ClusterClass vornehmen und ein paralleles Update von Clusterknoten mit benutzerdefinierter ClusterClass initiieren.

Hinweis Der im folgenden Workflow gezeigte Vorgang ist ein Beispiel dafür, welche Möglichkeiten Sie mit einer benutzerdefinierten ClusterClass haben. Ihre Anwendungsfälle können davon abweichen, aber der allgemeine Workflow sollte anwendbar sein.

Schritt	Aufgabe	Anleitung
3	Melden Sie sich über SSH bei einem der Worker-Knoten an, um zu bestätigen, dass Pakete aktualisiert werden müssen.	3: Bestätigen des Vorhandenseins von Paketaktualisierungen
4	Aktualisieren Sie die benutzerdefinierte ClusterClass mit einem neuen Befehl, der die Aktualisierungen durchführt.	4: Aktualisieren der benutzerdefinierten ClusterClass
5	Bestätigen Sie das Rollout neuer Knoten mit den bereits ausgeführten Updates.	5: Überprüfen des parallelen Updates von Clusterknoten

1: Erstellen einer benutzerdefinierten ClusterClass

Der erste Teil umfasst das Erstellen einer benutzerdefinierten ClusterClass mit dem Namen `ccc` (Abkürzung für `customclusterclass`), indem Sie die standardmäßige ClusterClass mit dem Namen `tanzukubernetescluster` klonen.

Hinweis Der benutzerdefinierte ClusterClass-Name ist benutzerdefiniert. Wenn Sie einen anderen Namen verwenden, passen Sie die Anweisungen entsprechend an.

1 Erstellen und konfigurieren Sie einen vSphere-Namespace mit dem Namen *ccc-ns*.

Konfigurieren Sie Berechtigungen, Speicher, Inhaltsbibliothek und VM-Klassen. Weitere Informationen finden Sie je nach Bedarf in der Kapitel 6 Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern.

Hinweis Der vSphere-Namespace-Name ist benutzerdefiniert. Wenn Sie einen anderen Namen verwenden, passen Sie die Anweisungen entsprechend an.

2 Melden Sie sich bei Supervisor an.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USER@vsphere.local
```

3 Schreiben Sie die Ausgabe der standardmäßigen ClusterClass in eine Datei mit dem Namen *ccc.yaml*.

```
kubectl -n ccc-ns get clusterclass tanzukubernetescluster -o yaml > ccc.yaml
```

Oder in abgekürzter Version:

```
kubectl -n ccc-ns get cc tanzukubernetescluster -o yaml > ccc.yaml
```

4 Öffnen Sie die geklonte ClusterClass-Datei zum Bearbeiten.

```
vim ccc.yaml
```

5 Bearbeiten Sie die Datei `ccc.yaml`.

- Löschen Sie die Zeile `metadata.creationTimestamp`.
- Löschen Sie die Zeile `metadata.generation`.
- Löschen Sie die Zeile `metadata.resourceVersion`.
- Löschen Sie die Zeile `metadata.uid`.
- Ändern Sie den Wert für `metadata.name` von `tanzukubernetescluster` in `ccc`.
- Ändern Sie den Wert `metadata.namespace` nicht: `ccc-ns`
- Behalten Sie den Wert `metadata.annotations` für `run.tanzu.vmware.com/resolve-tkr: ""` bei. Diese Anmerkung ist für die TKR-Daten/-Auflösung erforderlich.

6 Speichern und verifizieren Sie die Änderungen.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: ClusterClass
metadata:
  annotations:
    run.tanzu.vmware.com/resolve-tkr: ""
  name: ccc
  namespace: ccc-ns
spec:
...
```

7 Erstellen Sie das benutzerdefinierte ClusterClass-Objekt.

```
kubectl apply -f ccc.yaml -n ccc-ns
```

Erwartetes Ergebnis:

```
clusterclass.cluster.x-k8s.io/ccc created
```

8 Listen Sie die benutzerdefinierte ClusterClass auf.

```
kubectl get cc -n ccc-ns
```

Erwartetes Ergebnis:

```
NAME                      AGE
ccc                       3m14s
tanzukubernetescluster    29m
```

2: Erstellen eines TKG-Clusters basierend auf der benutzerdefinierten ClusterClass

Verwenden Sie die v1beta1-API des Clusters, um einen Cluster basierend auf einer ClusterClass zu erstellen.

1 Erstellen Sie das `ccc-cluster.yaml`-Manifest, um den Cluster bereitzustellen.

```
#ccc-cluster.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: ccc-cluster
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.0.2.0/16
    services:
      cidrBlocks:
      - 198.51.100.0/12
    serviceDomain: cluster.local
  topology:
```

```
    class: ccc
    version: v1.26.5---vmware.2-fips.1-tkg.1
    controlPlane:
      replicas: 1
    workers:
      machineDeployments:
        - class: node-pool
          name: tkgs-node-pool-1
          replicas: 1
    variables:
    - name: vmClass
      value: guaranteed-small
    - name: storageClass
      value: tkg-storage-profile
```

Dabei gilt:

- Der Wert für `metadata.name` ist der Name des Clusters: `ccc-cluster`
- Der Wert für `spec.topology.class` stimmt mit dem Namen der benutzerdefinierten ClusterClass überein: `ccc`
- Der Wert für `spec.topology.version` ist die TKR-Version
- Der Wert für `spec.topology.variables.storageClass` ist der Name der dauerhaften Speicherklasse

Hinweis Zu Testzwecken reicht 1 Replikat für die Steuerungsebene und den Worker-Knotenpool aus. Verwenden Sie in der Produktion 3 Replikate für die Steuerungsebene und mindestens 3 Replikate für jeden Worker-Knotenpool.

2 Erstellen Sie den TKG-Cluster basierend auf der benutzerdefinierten ClusterClass.

```
kubectl apply -f ccc-cluster.yaml -n ccc-ns
```

Erwartetes Ergebnis:

```
cluster.cluster.x-k8s.io/ccc-cluster created
```

3 Überprüfen Sie die Clusterbereitstellung.

Führen Sie den folgenden Befehl aus. Warten Sie, bis alle Clusterknoten ordnungsgemäß hochgefahren wurden.

```
kubectl -n ccc-ns get
cc,clusters,vsphereclusters,kcp,machinedeployment,machineset,machine,vspheremachine,virtual
machineservice
```

Hinweis Es ist hilfreich, diesen Befehl in einer separaten Sitzung auszuführen, damit Sie den Fortschritt des parallelen Updates in Schritt 5 überwachen können.

3: Bestätigen des Vorhandenseins von Paketaktualisierungen

Melden Sie sich über SSH bei einem der Worker-Knoten an, um zu bestätigen, dass Pakete aktualisiert werden müssen.

Hinweis Das Ziel bei diesem Schritt besteht einfach darin, zu bestätigen, dass zu aktualisierende Pakete vorhanden sind, und nicht darin, sie tatsächlich zu aktualisieren. Sie werden von der benutzerdefinierten ClusterClass aktualisiert, wenn neue Clusterknoten bereitgestellt werden (folgende Schritte). Dieser und die folgenden Schritte sollen ein Beispiel dafür sein, welche Möglichkeiten Sie mit einer benutzerdefinierten ClusterClass haben.

1 Führen Sie den folgenden Befehl aus, um den geheimen SSH-Schlüssel abzurufen.

```
export CC=ccc-cluster && kubectl get secret -n ccc-ns ${CC}-ssh -o jsonpath={.data.ssh-privatekey} | base64 -d > ${CC}-ssh && chomd 4000 ${CC}-ssh
```

2 Führen Sie den folgenden Befehl aus, um die IP-Adresse der Worker-Knoten-VM abzurufen.

```
kubectl -n ccc-ns get vm -o wide
```

Hinweis Wenn Sie mehrere Worker-Knoten bereitgestellt haben, wählen Sie einen davon aus. Verwenden Sie keinen Steuerungsebenenknoten.

3 Führen Sie den folgenden Befehl aus, um sich per SSH bei der Worker-Knoten-VM anzumelden.

```
ssh -i ${CC}-ssh vmware-system-user@IP-ADDRESS-OF-WORKER-NODE
```

Beispiel:

```
ssh -i ${CC}-ssh vmware-system-user@192.168.128.55
```

Hinweis Geben Sie „Ja“ ein, um mit dem Herstellen der Verbindung fortzufahren.

Erwartetes Ergebnis: Nach dem Anmelden per SSH beim Host sollte die folgende Meldung angezeigt werden.

```
tdnf update info not availble yet!
```

4 Führen Sie die folgenden Befehle aus und suchen Sie nach Updates.

```
sudo -i
```

```
tdnf update
```

5 Geben Sie an der Eingabeaufforderung „N“ für „Nein“ ein (nicht aktualisieren).

Erwartetes Ergebnis:

```
Operation aborted
```

Hinweis Der Zweck besteht hier einfach darin, zu überprüfen, ob Aktualisierungen vorhanden sind, und nicht darin, Aktualisierungen zu initiieren. Sie initiieren die Aktualisierungen, indem Sie im nächsten Abschnitt einen Befehl zur benutzerdefinierten ClusterClass hinzufügen.

6 Geben Sie „exit“ ein, um sich von der SSH-Sitzung abzumelden, und geben Sie dann erneut „exit“ ein.

4: Aktualisieren der benutzerdefinierten ClusterClass

Aktualisieren Sie die benutzerdefinierte ClusterClass mit einem neuen Befehl, der ein tdnf-Update durchführt.

1 Öffnen Sie die benutzerdefinierte ClusterClass mit dem Namen `ccc`.

```
kubectl edit cc ccc -n ccc-ns
```

2 Scrollen Sie zum folgenden Abschnitt mit `postKubeadmCommands`.

```
  - definitions:
    - jsonPatches:
      - op: add
        path: /spec/template/spec/kubeadmConfigSpec/postKubeadmCommands
        valueFrom:
          template: |
            - touch /root/kubeadm-complete
            - vmware-rpctool 'info-set guestinfo.kubeadm.phase complete'
            - vmware-rpctool 'info-set guestinfo.kubeadm.error ---'
      selector:
        apiVersion: controlplane.cluster.x-k8s.io/v1beta1
        kind: KubeadmControlPlaneTemplate
        matchResources:
          controlPlane: true
    - jsonPatches:
      - op: add
        path: /spec/template/spec/postKubeadmCommands
        valueFrom:
          template: |
            - touch /root/kubeadm-complete
            - vmware-rpctool 'info-set guestinfo.kubeadm.phase complete'
            - vmware-rpctool 'info-set guestinfo.kubeadm.error ---'
      selector:
        apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
        kind: KubeadmConfigTemplate
        matchResources:
```

```
          machineDeploymentClass:
            names:
            - node-pool
    name: controlPlanePostKubeadmCommandsSuccess
```

Fügen Sie beiden `valueFrom.template` Feldern den folgenden Befehl hinzu.

```
- tdnf update -y
```

Beispiel:

```
  - definitions:
    - jsonPatches:
      - op: add
        path: /spec/template/spec/kubeadmConfigSpec/postKubeadmCommands
        valueFrom:
          template: |
            - touch /root/kubeadm-complete
            - vmware-rpctool 'info-set guestinfo.kubeadm.phase complete'
            - vmware-rpctool 'info-set guestinfo.kubeadm.error ---'
            - tdnf update -y
      selector:
        apiVersion: controlplane.cluster.x-k8s.io/v1beta1
        kind: KubeadmControlPlaneTemplate
        matchResources:
          controlPlane: true
    - jsonPatches:
      - op: add
        path: /spec/template/spec/postKubeadmCommands
        valueFrom:
          template: |
            - touch /root/kubeadm-complete
            - vmware-rpctool 'info-set guestinfo.kubeadm.phase complete'
            - vmware-rpctool 'info-set guestinfo.kubeadm.error ---'
            - tdnf update -y
      selector:
        apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
        kind: KubeadmConfigTemplate
        matchResources:
          machineDeploymentClass:
            names:
            - node-pool
    name: controlPlanePostKubeadmCommandsSuccess
```

3 Speichern Sie die Änderungen an der benutzerdefinierten ClusterClass und schließen Sie den Editor.

```
wq
```

Erwartetes Ergebnis:

```
clusterclass.cluster.x-k8s/ccc edited
```

5: Überprüfen des parallelen Updates von Clusterknoten

Durch das Aktualisieren der benutzerdefinierten ClusterClass wird ein paralleles Update von Clusterknoten für Cluster, die basierend auf dieser ClusterClass bereitgestellt werden, ausgelöst. Die neuen Knoten werden hochgefahren, nachdem der obige Befehl bereits auf sie angewendet wurde.

1 Überprüfen Sie die Clusterbereitstellung, indem Sie folgenden Befehl ausführen:

   Warten Sie, bis alle Clusterknoten ordnungsgemäß hochgefahren wurden.

   ```
   kubectl -n ccc-ns get
   cc,clusters,vsphereclusters,kcp,machinedeployment,machineset,machine,vspheremachine,virtual
   machineservice
   ```

2 Es sollte zu sehen sein, dass neue Knoten mit neuen UUIDs bereitgestellt werden.

3 Führen Sie den folgenden Befehl aus, um sich per SSH bei der Worker-Knoten-VM anzumelden.

   ```
   ssh -i ${CC}-ssh vmware-system-user@IP-ADDRESS-OF-WORKER-NODE
   ```

   Erwartetes Ergebnis: Nach dem Anmelden per SSH beim Host sollte die folgende Meldung angezeigt werden.

   ```
   tdnf update info not availble yet!
   ```

4 Führen Sie die folgenden Befehle aus:

   ```
   sudo -i
   ```

   ```
   tdnf update
   ```

   Erwartetes Ergebnis: Es sollten viel weniger zu aktualisierende Pakete angezeigt werden.

5 Geben Sie an der Eingabeaufforderung „N“ für „Nein“ ein (nicht aktualisieren).

   Erwartetes Ergebnis:

   ```
   Operation aborted
   ```

6 Führen Sie den folgenden Befehl aus, um zu bestätigen, dass tdnf ausgeführt wurde.

   ```
   cat /var/log/cloud-init-output.log | grep -i tdnf
   ```

7 Geben Sie „exit“ ein, um sich von der SSH-Sitzung abzumelden, und geben Sie dann erneut „exit“ ein.

Warten einer benutzerdefinierten ClusterClass

Nach dem Upgrade des TKG-Diensts auf eine neue Version müssen Sie sicherstellen, dass Ihre benutzerdefinierte ClusterClass, die aus der Standard-ClusterClass der vorherigen TKG-Dienstversion abgeleitet wurde, mit den Änderungen an der Standard-ClusterClass aktualisiert wird, die mit der neuen TKG-Dienstversion ausgeliefert wird.

Verwenden Sie den folgenden Workflow, um Ihre benutzerdefinierte ClusterClass mit der vom System bereitgestellten ClusterClass synchron zu halten. Beachten Sie, dass bei diesen Anweisungen davon ausgegangen wird, dass Sie eine anfängliche benutzerdefinierte ClusterClass erstellt haben, wie hier beschrieben.

1 Upgrade der TKG-Dienstversion

   Führen Sie beispielsweise ein Upgrade von TKG-Dienst v3.0 auf v3.1 durch.

   Weitere Informationen hierzu finden Sie unter Kapitel 3 Installieren und Aktualisieren des TKG-Diensts.

2 Erstellen Sie eine neue benutzerdefinierte ClusterClass, indem Sie den hier angegebenen Workflows auf hoher Ebene folgen.

   Versionieren Sie die neue benutzerdefinierte ClusterClass manuell, indem Sie an ihren Namen die TKG-Dienstversion anhängen, etwa `ccc-3.1`.

3 Fügen Sie der neuen benutzerdefinierten ClusterClass die benutzerdefinierten Patches und Variablen aus der vorherigen benutzerdefinierten ClusterClass hinzu.

   Verwenden Sie dazu `cat ccc.yaml`, und kopieren Sie die benutzerdefinierten Patches und Variablen daraus nach `ccc-3.1.yaml`.

4 Wenden Sie die neue benutzerdefinierte ClusterClass an, und warten Sie, bis der Abgleich erfolgreich verläuft.

5 Aktualisieren Sie TKG-Cluster mit der vorherigen benutzerdefinierten ClusterClass auf die neue benutzerdefinierte ClusterClass, indem Sie das Feld `spec.topology.class` im Cluster-Objekt bearbeiten.

Nicht verwaltete ClusterClass

Für vSphere 8 U2+ gibt es eine Anmerkung, die Sie zu einer benutzerdefinierten ClusterClass hinzufügen können, wenn der TKG-Controller die benutzerdefinierte ClusterClass nicht verwalten soll. Beachten Sie, dass Sie für die manuelle Erstellung aller zugrunde liegenden Kubernetes-Objekte wie Zertifikate, geheime Schlüssel usw. verantwortlich sind, wenn Sie diese Anmerkung hinzufügen. Weitere Informationen hierzu finden Sie in der v1beta1-Beispiel: Cluster basierend auf einer benutzerdefinierten ClusterClass (vSphere 8 U1-Workflow) zur benutzerdefinierten vSphere 8 U1-ClusterClass.

Die Anmerkung lautet folgendermaßen:

Anmerkungsschlüssel	Wert
run.tanzu.vmware.com/unmanaged-clusterclass	" "

Im Folgenden finden Sie ein Beispiel zum Hinzufügen der Anmerkung zur benutzerdefinierten ClusterClass mit der Bezeichnung `ccc`:

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: ClusterClass
metadata:
  annotations:
    run.tanzu.vmware.com/resolve-tkr: ""
    run.tanzu.vmware.com/unmanaged-clusterclass: ""
  name: ccc
  namespace: ccc-ns
spec:
...
```

v1beta1-Beispiel: Cluster basierend auf einer benutzerdefinierten ClusterClass (vSphere 8 U1-Workflow)

Lesen Sie diese Anweisungen, um einen TKG-Cluster basierend auf einer benutzerdefinierten ClusterClass bereitzustellen. Beachten Sie, dass diese Anweisungen spezifisch für vSphere 8 U1-Umgebungen gelten.

Voraussetzungen

Das Verfahren für die Bereitstellung eines TKG-Clusters basierend auf einer benutzerdefinierten ClusterClass ist ab Version vSphere 8 U1 verfügbar. Wenn Sie vSphere 8 U2 verwenden, finden Sie weitere Informationen unter v1beta1-Beispiel: Cluster basierend auf einer benutzerdefinierten ClusterClass (vSphere 8 U2- und höherer Workflow).

Beachten Sie die folgenden Voraussetzungen.

- vSphere 8 U1-Umgebung
- Arbeitslastverwaltung aktiviert
- Supervisor konfiguriert
- Ubuntu-Client mit installierten Kubernetes-CLI-Tools für vSphere

Achtung Die benutzerdefinierte ClusterClass ist eine experimentelle Kubernetes-Funktion gemäß der Upstream-Cluster-API-Dokumentation. Aufgrund der im Zusammenhang mit der benutzerdefinierten ClusterClass verfügbaren Bandbreite an Anpassungen kann VMware nicht alle möglichen Anpassungen testen oder validieren. Die Kunden sind für das Testen, Validieren und Beheben von Fehlern ihrer benutzerdefinierten ClusterClass-Cluster verantwortlich. Sie können Support-Tickets für ihre benutzerdefinierten ClusterClass-Cluster öffnen. VMware Support beschränkt sich jedoch auf eine Unterstützung nach bestem Wissen und kann nicht garantieren, dass alle Probleme behoben werden, die bei benutzerdefinierten ClusterClass-Clustern auftreten. Die Kunden sollten sich diese Risiken bewusst machen, bevor sie benutzerdefinierte ClusterClass-Cluster in Produktionsumgebungen bereitstellen.

Teil 1: Erstellen der benutzerdefinierten ClusterClass

Der erste Teil umfasst das Erstellen einer benutzerdefinierten ClusterClass, indem Sie die standardmäßige ClusterClass mit dem Namen `tanzukubernetescluster` klonen.

1 Erstellen Sie einen vSphere-Namespace mit dem Namen *custom-ns*.

2 Melden Sie sich bei Supervisor an.

3 Führen Sie einen Kontextwechsel zum vSphere-Namespace mit dem Namen *custom-ns* durch.

4 Rufen Sie die standardmäßige ClusterClass ab.

```
kubectl get clusterclass tanzukubernetescluster -o json
```

5 Erstellen Sie eine benutzerdefinierte ClusterClass mit dem Namen *custom-cc*, indem Sie die standardmäßige ClusterClass klonen.

```
kubectl get clusterclass tanzukubernetescluster -o json | jq '.metadata.name="custom-cc"'
| kubectl apply -f -
```

Erwartetes Ergebnis:

```
clusterclass.cluster.x-k8s.io/custom-cc created
```

6 Rufen Sie die benutzerdefinierte ClusterClass ab.

```
kubectl get clusterclass custom-cc -o json
```

Bei Bedarf können Sie „less“ verwenden, um die benutzerdefinierte ClusterClass anzuzeigen.

```
kubectl get clusterclass custom-cc -o json | less
```

Hinweis Führen Sie den Befehl „q“ aus, um „less“ zu beenden.

Teil 2: Erstellen erforderlicher Supervisor-Objekte für die Bereitstellung des TKG-Clusters

Im nächsten Teil geht es um die Erstellung der Supervisor-Objekte, die für die anfängliche Bereitstellung eines benutzerdefinierten TKG-Clusters mithilfe der benutzerdefinierten ClusterClass erforderlich sind.

Hinweis Standardmäßig wird der Clustername „ccc-cluster“ verwendet. Falls Sie einen anderen Clusternamen verwenden, müssen Sie die entsprechenden Felder ändern.

1 Erstellen Sie den Aussteller für das selbstsignierte Erweiterungszertifikat.

```
#self-signed-extensions-issuer.yaml
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: self-signed-extensions-issuer
spec:
  selfSigned: {}
```

```
kubectl apply -f self-signed-extensions-issuer.yaml -n custom-ns
```

Erwartetes Ergebnis:

```
issuer.cert-manager.io/self-signed-extensions-issuer created
```

2 Erstellen Sie den geheimen Schlüssel für das CA-Zertifikat für Erweiterungen.

```
#extensions-ca-certificate.yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ccc-cluster-extensions-ca
spec:
  commonName: kubernetes-extensions
  duration: 87600h0m0s
  isCA: true
  issuerRef:
    kind: Issuer
    name: self-signed-extensions-issuer
  secretName: ccc-cluster-extensions-ca
  usages:
  - digital signature
  - cert sign
  - crl sign
```

```
kubectl apply -f extensions-ca-certificate.yaml -n custom-ns
```

Erwartetes Ergebnis:

```
certificate.cert-manager.io/ccc-cluster-extensions-ca created
```

3 Erstellen Sie den Aussteller für das CA-Zertifikat für Erweiterungen.

```
#extensions-ca-issuer.yaml
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: ccc-cluster-extensions-ca-issuer
spec:
  ca:
    secretName: ccc-cluster-extensions-ca
```

```
kubectl apply -f extensions-ca-issuer.yaml -n custom-ns
```

Erwartetes Ergebnis:

```
issuer.cert-manager.io/ccc-cluster-extensions-ca-issuer created
```

4 Erstellen Sie den Geheimschlüssel für das Zertifikat des Authentifizierungsdiensts.

```
#auth-svc-cert.yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ccc-cluster-auth-svc-cert
spec:
  commonName: authsvc
  dnsNames:
  - authsvc
  - localhost
  - 127.0.0.1
  duration: 87600h0m0s
  issuerRef:
    kind: Issuer
    name: ccc-cluster-extensions-ca-issuer
  secretName: ccc-cluster-auth-svc-cert
  usages:
  - server auth
  - digital signature
```

```
kubectl apply -f auth-svc-cert.yaml -n custom-ns
```

Erwartetes Ergebnis:

```
certificate.cert-manager.io/ccc-cluster-auth-svc-cert created
```

5 Überprüfen Sie die Erstellung der Aussteller und Zertifikate.

```
kubectl get issuers -n custom-ns
NAME                                READY   AGE
ccc-cluster-extensions-ca-issuer    True    2m57s
self-signed-extensions-issuer       True    14m
```

```
kubectl get certs -n custom-ns
NAME                          READY   SECRET                        AGE
ccc-cluster-auth-svc-cert     True    ccc-cluster-auth-svc-cert     34s
ccc-cluster-extensions-ca     True    ccc-cluster-extensions-ca     5m
```

Teil 3: Erstellen eines TKG-Clusters basierend auf der benutzerdefinierten ClusterClass

Sie verwenden die v1beta1-API des Clusters, um einen Cluster basierend auf einer ClusterClass zu erstellen. Ein v1beta1-Cluster, der auf einer benutzerdefinierten ClusterClass basiert, erfordert das folgende Minimum an Variablen:

Variable	Beschreibung
vmClass	Weitere Informationen hierzu finden Sie unter Verwenden von VM-Klassen mit TKG-Dienstclustern.
storageClass	Weitere Informationen hierzu finden Sie unter Konfigurieren eines dauerhaften Speichers für den vSphere-Namespace.
ntp	NTP-Server, der zum Aktivieren von Supervisor verwendet wird.
extensionCert	Wird automatisch generiert, nachdem das „Erweiterungs-CA-Zertifikat“ im vorherigen Abschnitt erstellt wurde.
clusterEncryptionConfigYaml	Im Abschnitt unten wird der Vorgang zum Abrufen dieser Datei Schritt für Schritt erläutert.

1 Erstellen Sie den geheimen Verschlüsselungsschlüssel.

```
#encryption-secret.yaml
apiVersion: v1
data:
  key: all3dzZpODFmRmh6MVlJbUtQQktuN2ViQzREbDBQRHlxVk8yYXRxTW9QQT0=
kind: Secret
metadata:
  name: ccc-cluster-encryption
type: Opaque
```

```
kubectl apply -f encryption-secret.yaml -n custom-ns
```

Erwartetes Ergebnis:

```
secret/ccc-cluster-encryption created
```

2 Erfassen Sie den NTP-Server aus Supervisor.

```
kubectl -n vmware-system-vmop get configmap vmoperator-network-config -o
jsonpath={.data.ntpservers}
```

3 Erstellen Sie das `cluster-with-ccc.yaml`-Manifest, um den Cluster bereitzustellen.

```
#cluster-with-ccc.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: ccc-cluster
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 193.0.0.0/16
    serviceDomain: managedcluster1.local
    services:
      cidrBlocks:
      - 198.201.0.0/16
  topology:
    class: custom-cc
    version: v1.26.5---vmware.2-fips.1-tkg.1
    controlPlane:
      metadata: {}
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          metadata: { }
          name: node-pool-workers
          replicas: 3
    variables:
    - name: vmClass
      value: guaranteed-medium
    - name: storageClass
      value: tkg-storage-profile
    - name: ntp
      value: time.acme.com
    - name: extensionCert
      value:
        contentSecret:
          key: tls.crt
          name: ccc-cluster-extensions-ca
    - name: clusterEncryptionConfigYaml
      value: LS0tCm...Ht9Cg==
```

Überprüfen oder aktualisieren Sie im Cluster-Manifest die folgenden Felder:

Parameter	Beschreibung
`metadata.name`	Name des v1beta1-Clusters.
`spec.topology.class`	Name der benutzerdefinierten ClusterClass.

Parameter	Beschreibung
`spec.topology.version`	Tanzu Kubernetes-Version-Version
`spec.topology.variables.storageClass.value`	Speicherrichtlinie, die an den vSphere-Namespace angehängt ist, in dem der Cluster bereitgestellt wird
`spec.topology.variables.ntp.value`	NTP-Serveradresse
`spec.topology.variables.extensionCert.value.contentSecret.name`	Überprüfen
`spec.topology.variables.clusterEncryptionConfigYaml.value`	Füllen Sie dieses Feld mit dem `data.key`-Wert aus dem geheimen ClusterEncryptionConfig-Schlüssel auf.

4 Erstellen Sie den Cluster basierend auf der benutzerdefinierten ClusterClass.

```
kubectl apply -f cluster-with-ccc.yaml -n custom-ns
```

Erwartetes Ergebnis:

```
cluster.cluster.x-k8s.io/ccc-cluster created
```

Stellen Sie mithilfe des vSphere Clients sicher, dass der Cluster erstellt wurde.

5 Melden Sie sich beim TKG-Cluster an.

```
kubectl vsphere login --server=xxx.xxx.xxx.xxx --vsphere-username USERNAME@vsphere.local
--tanzu-kubernetes-cluster-name ccc-cluster --tanzu-kubernetes-cluster-namespace custom-ns
```

Teil 4: Erstellen erforderlicher Supervisor-Objekte zum Verwalten des TKG-Clusters

Sobald der Cluster mit CCC angewendet wurde, versuchen verschiedene Controller, ihn bereitzustellen. Die zugrunde liegenden Infrastrukturressourcen erfordern jedoch weiterhin ein ordnungsgemäßes Bootstrapping zusätzlicher Objekte.

Parameter	Wert
Authentifizierung	Authentifizierungswerte müssen erfasst und in einer Datei mit dem Namen „values.yaml“ aktualisiert werden
Base64-verschlüsselt	Die values.yaml-Datei wird in eine base64-Zeichenfolge codiert.
guest-cluster-auth-service-data-values.yaml	Diese Zeichenfolge wird der guest-cluster-auth-service-data-values.yaml-Datei hinzugefügt, die vor der Anwendung der Datei aus CCC_config_yamls.tar.gz heruntergeladen wurde.
Geheimer GuestClusterAuthSvcDataValues-Schlüssel	Schließlich muss das Gastcluster-Bootstrap so geändert werden, dass es auf den neu erstellten geheimen GuestClusterAuthSvcDataValues-Schlüssel verweist.

1 Wechseln Sie den Kontext zum vSphere-Namespace, in dem der Cluster bereitgestellt wird.

```
kubectl config use-context custom-ns
```

2 Rufen Sie den Wert von `authServicePublicKeys` ab.

```
kubectl -n vmware-system-capw get configmap vc-public-keys -o
jsonpath="{.data.vsphere\.local\.json}"
```

Kopieren Sie das Ergebnis in eine Textdatei mit dem Namen `values.yaml`.

```
authServicePublicKeys: '{"issuer_url":"https://...SShrDw=="]}]}}'
```

3 Rufen Sie die Cluster-UID ab, um `authServicePublicKeys` zu aktualisieren.

```
kubectl get cluster -n custom-ns ccc-cluster -o yaml | grep uid
```

4 Hängen Sie im Abschnitt `authServicePublicKeys` der `values.yaml`-Datei die Cluster-UID an den Wert „`client_id`“ an.

Syntax: `vmware-tes:vc:vns:k8s:`*`clusterUID`*

Beispiel:

```
vmware-tes:vc:vns:k8s:7d95b50b-4fd4-4642-82a3-5dbfe87f499c
```

5 Rufen Sie den Zertifikatswert ab (ersetzen Sie *ccc-cluster* durch den ausgewählten Clusternamen).

```
kubectl -n custom-ns get secret ccc-cluster-auth-svc-cert -o jsonpath="{.data.tls\.crt}" |
base64 -d
```

6 Fügen Sie das Zertifikat in der Datei `values.yaml` hinzu.

Fügen Sie den Zertifikatsinhalt unterhalb des Abschnitts `authServicePublicKeys` hinzu.

Hinweis Das Zertifikat muss zur Vermeidung von Fehlern um 4 Leerzeichen eingerückt werden.

Beispiel:

```
authServicePublicKeys: '{"issuer_url":"https://...SShrDw=="]}]}}'
ceritificate: |
    -----BEGIN CERTIFICATE-----
    MIIDPTCCAiWgAwIBAgIQMibGSjeuJelQoPxCof/+xzANBgkqhkiG9w0BAQsFADAg
    ...
    sESk/RDTB1UAvi8PD3zcbEKZuRxuo4IAJqFFbAabwULhjUo0UwT+dIJo1gLf5/ep
    VoIRJS7j6VT98WbKyZp5B4I=
    -----END CERTIFICATE-----
```

7 Rufen Sie den privateKey-Wert ab.

```
kubectl -n custom-ns get secret ccc-cluster-auth-svc-cert -o jsonpath="{.data.tls\.key}"
```

8 Überprüfen Sie Ihre `values.yaml`-Datei.

```
authServicePublicKeys: '{"issuer_url":"https://10.197.79.141/openidconnect/
vsphere.local","client_id":"vmware-tes:vc:vns:k8s:7d95...499c",...SShrDw=="]}]}}'
certificate: |
    -----BEGIN CERTIFICATE-----
    MIIDPTCCAiWgAwIBAgIQWQyXAQDRMhgrGre8ysVN0DANBgkqhkiG9w0BAQsFADAg
    ...
    uJSBP49sF0nKz5nf7w+BdYE=
    -----END CERTIFICATE-----
privateKey: LS0tLS1CRUdJTi...VktLS0tLQo=
```

9 Hashen Sie die `values.yaml`-Datei mit Base64-Codierung, um die Ausgabe für die `guest-cluster-auth-service-data-values.yaml`-Datei zu erfassen.

```
base64 -i values.yaml -w 0
```

10 Erstellen Sie die `guest-cluster-auth-service-data-values.yaml`-Datei.

Nachstehend finden Sie eine Vorlage für den geheimen Schlüssel.

```
apiVersion: v1
data:
  values.yaml: YXV0a...ExRbz0K
kind: Secret
metadata:
  labels:
    tkg.tanzu.vmware.com/cluster-name: ccc-cluster
    tkg.tanzu.vmware.com/package-name: guest-cluster-auth-
service.tanzu.vmware.com.1.3.0+tkg.2-vmware
  name: ccc-cluster-guest-cluster-auth-service-data-values
type: Opaque
```

Informationen zum Auffüllen der erwarteten Werte des geheimen Schlüssels finden Sie in der folgenden Tabelle.

Parameter	Wert
`data.values.yaml`	Base64-codierte Zeichenfolge von `values.yaml`
`metadata.labels.cluster-name`	Name des Clusters, wie z. B. `ccc-cluster`
`metadata.labels.package-name`	guest-cluster-auth-service.tanzu.vmware.com.*version* Führen Sie zum Abrufen dieses Werts den folgenden Befehl aus: `kubectl get tkr v1.26.5---vmware.2-fips.1-tkg.1 -o yaml` Ändern Sie die TKR-Version je nach der von Ihnen verwendeten Version
`metadata.name`	Name des Clusters, wie z. B. `ccc-cluster`

11 Erstellen Sie den geheimen Schlüssel `guest-cluster-auth-service-data-values.yaml`.

```
kubectl apply -f guest-cluster-auth-service-data-values.yaml -n custom-ns
```

12 Bearbeiten Sie das Cluster-Bootstrap so, dass es auf den geheimen Schlüssel verweist.

```
kubectl edit clusterbootstrap ccc-cluster -n custom-ns
```

13 Fügen Sie die folgenden Zeilen unterhalb der Zeile `guest-cluster-auth-service.tanzu.vmware.com.version:` hinzu.

```
valuesFrom:
  secretRef: ccc-cluster-guest-cluster-auth-service-data-values
```

Beispiel:

```
spec:
  additionalPackages:
  - refName: guest-cluster-auth-service.tanzu.vmware.com.1.3.0+tkg.2-vmware
    valuesFrom:
      secretRef: ccc-cluster-guest-cluster-auth-service-data-values
```

14 Speichern und beenden Sie, um die clusterbootstrap-Änderungen anzuwenden.

Teil 5: Konfigurieren der Pod-Sicherheit

Wenn Sie die TKR-Version 1.25 oder höhere Versionen verwenden, konfigurieren Sie die Pod-Sicherheit für den vSphere-Namespace mit dem Namen `custom-ns`. Weitere Informationen hierzu finden Sie unter Konfigurieren von PSA für TKR 1.25 und höher.

Wenn Sie die TKR-Version 1.24 oder frühere Versionen verwenden, müssen Pods im Cluster an die Pod-Sicherheitsrichtlinie gebunden werden. Gehen Sie wie folgt vor, um die erforderlichen Ressourcenobjekte auf Clusterebene anzuwenden.

1 Erfassen Sie die kubeconfig des TKG-Clusters.

```
kubectl -n custom-ns get secret ccc-cluster-kubeconfig -o jsonpath="{.data.value}" |
base64 -d > ccc-cluster-kubeconfig
```

2 Erstellen Sie die `psp.yaml`-Datei.

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: tanzu-system-kapp-ctrl-restricted
spec:
  privileged: false
  allowPrivilegeEscalation: false
  requiredDropCapabilities:
    - ALL
  volumes:
    - configMap
    - emptyDir
    - projected
    - secret
    - downwardAPI
```

```
    - persistentVolumeClaim
  hostNetwork: false
  hostIPC: false
  hostPID: false
  runAsUser:
    rule: MustRunAsNonRoot
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: MustRunAs
    ranges:
      - min: 1
        max: 65535
  fsGroup:
    rule: MustRunAs
    ranges:
      - min: 1
        max: 65535
  readOnlyRootFilesystem: false
```

3 Wenden Sie die Pod-Sicherheitsrichtlinie an.

```
KUBECONFIG=ccc-cluster-kubeconfig kubectl apply -f psp.yaml
```

4 Melden Sie sich beim TKG-Cluster an.

```
kubectl vsphere login --server=10.197.154.66 --vsphere-username
administrator@vsphere.local --insecure-skip-tls-verify --tanzu-kubernetes-cluster-name ccc-
cluster --tanzu-kubernetes-cluster-namespace custom-ns
```

5 Listen Sie die Namespaces auf.

```
KUBECONFIG=ccc-cluster-kubeconfig kubectl get ns -A
```

```
NAME                            STATUS   AGE
default                         Active   13d
kube-node-lease                 Active   13d
kube-public                     Active   13d
kube-system                     Active   13d
secretgen-controller            Active   13d
tkg-system                      Active   13d
vmware-system-antrea            Active   13d
vmware-system-cloud-provider    Active   13d
vmware-system-csi               Active   13d
vmware-system-tkg               Active   13d
```

Teil 6: Synchronisieren vSphere SSO-Rollen mit dem benutzerdefinierten TKG-Cluster

Das RoleBinding für vCenter Single Sign-On-Benutzer, das in die vSphere-Namespaces integriert ist, muss aus Supervisor in den TKG-Cluster synchronisiert werden, damit Entwickler die Cluster-Arbeitslasten verwalten können.

Dieser Vorgang erfordert den Export der vorhandenen RoleBinding-Liste aus Supervisor, das Erfassen von RoleBindings, die die Rolle „Bearbeiten“ aufweisen, das Erstellen der Datei `sync-cluster-edit-rolebinding.yaml` und das anschließende Anwenden auf den TKG-Cluster mithilfe seiner KUBECONFIG.

1 Erfassen Sie vorhandene RoleBindings aus Supervisor.

```
kubectl get rolebinding -n custom-ns  -o yaml
```

2 Identifizieren Sie in der zurückgegebenen Liste der RoleBinding-Objekte diejenigen, deren `roleRef.name` „edit“ lautet.

Beispiel:

```
apiVersion: v1
items:
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: "2023-08-25T18:44:45Z"
    name: ccc-cluster-8lr5x-ccm
    namespace: custom-ns
    ownerReferences:
    - apiVersion: vmware.infrastructure.cluster.x-k8s.io/v1beta1
      blockOwnerDeletion: true
      controller: true
      kind: ProviderServiceAccount
      name: ccc-cluster-8lr5x-ccm
      uid: b5fb9f01-9a55-4f69-8673-fadc49012994
    resourceVersion: "108766602"
    uid: eb93efd4-ae56-4d9f-a745-d2782885e7fb
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: Role
    name: ccc-cluster-8lr5x-ccm
  subjects:
  - kind: ServiceAccount
    name: ccc-cluster-8lr5x-ccm
    namespace: custom-ns
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: "2023-08-25T18:44:45Z"
    name: ccc-cluster-8lr5x-pvcsi
    namespace: custom-ns
    ownerReferences:
    - apiVersion: vmware.infrastructure.cluster.x-k8s.io/v1beta1
      blockOwnerDeletion: true
      controller: true
      kind: ProviderServiceAccount
      name: ccc-cluster-8lr5x-pvcsi
      uid: d9342f8f-13d2-496d-93cb-b24edfacb5c1
    resourceVersion: "108766608"
    uid: fd1820c7-7993-4299-abb7-bb67fb17f1fd
```

```
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: Role
    name: ccc-cluster-8lr5x-pvcsi
  subjects:
  - kind: ServiceAccount
    name: ccc-cluster-8lr5x-pvcsi
    namespace: custom-ns
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: "2023-08-25T16:58:06Z"
    labels:
      managedBy: vSphere
    name: wcp:custom-ns:group:vsphere.local:administrators
    namespace: custom-ns
    resourceVersion: "108714148"
    uid: d74a98c7-e7da-4d71-b1d5-deb60492d429
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: edit
  subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: sso:Administrators@vsphere.local
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: "2023-08-25T16:58:21Z"
    labels:
      managedBy: vSphere
    name: wcp:custom-ns:user:vsphere.local:administrator
    namespace: custom-ns
    resourceVersion: "108714283"
    uid: 07f7dbba-2670-4100-a59b-c09e4b2edd6b
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: edit
  subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: User
    name: sso:Administrator@vsphere.local
kind: List
metadata:
  resourceVersion: ""
```

3 Erstellen Sie eine Datei mit dem Namen `sync-cluster-edit-rolebinding.yaml`, um gegebenenfalls zusätzlich zu administrator@vsphere.local vorliegende RoleBindings hinzuzufügen. Beispiel:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    run.tanzu.vmware.com/vmware-system-synced-from-supervisor: "yes"
  name: vmware-system-auth-sync-wcp:custom-ns:group:vsphere.local:administrators
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: sso:Administrators@vsphere.local
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    run.tanzu.vmware.com/vmware-system-synced-from-supervisor: "yes"
  name: vmware-system-auth-sync-wcp:custom-ns:group:SSODOMAIN.COM:testuser
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: sso:testuser@SSODOMAIN.COM
```

Hinweis Im Feld `metadata.name` wird für alle Benutzer der Benutzerrolle `vmware-system-auth-sync-` vorangestellt. Die metadata.name- und subjects.name-Einträge müssen für alle nicht standardmäßigen Rollen geändert werden.

4 Wenden Sie die sync-cluster-edit-rolebinding.yaml-Konfiguration an, um Rollenbindungen zu synchronisieren.

```
KUBECONFIG=ccc-cluster-kubeconfig kubectl apply -f sync-cluster-edit-rolebinding.yaml
```

Verwenden der v1alpha3-API von TanzuKubernetesCluster

Dieser Abschnitt enthält Referenzinhalte für die Bereitstellung eines TKG-Dienstclusters vom Typ `TanzuKubernetesCluster` mithilfe der v1alpha3-API sowie Beispiele mit verschiedenen Konfigurationen und Anpassungen zur Erfüllung Ihrer Anforderungen.

TanzuKubernetesCluster v1alpha3-API

Mit der v1alpha3-API können Sie einen TanzuKubernetesCluster mithilfe von TKG auf Supervisor bereitstellen. In diesem Thema finden Sie die v1alpha3-API-Dokumentation.

TanzuKubernetesCluster v1alpha3-API

In der Spezifikation werden alle verfügbaren Parameter für die Bereitstellung eines `TanzuKubernetesCluster` mithilfe der v1alpha3-API aufgelistet.

Wichtig Ein gültiger Schlüsselname darf nur aus alphanumerischen Zeichen, einem Bindestrich (z. B. `key-name`), einem Unterstrich (z. B. `KEY_NAME`) oder einem Punkt (z. B. `key.name`) bestehen. Leerzeichen können in einem Schlüsselnamen nicht verwendet werden.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: string
  namespace: string
  annotations:
    run.tanzu.vmware.com/resolve-os-image: os-name=string
spec:
  topology:
    controlPlane:
      replicas: int32
      vmClass: string
      storageClass: string
      volumes:
        - name: string
          mountPath: string
          capacity:
            storage: size in GiB
      tkr:
        reference:
          name: string
      nodeDrainTimeout: string
    nodePools:
    - name: string
      failureDomain: string
      labels: map[string]string
      taints:
        -  key: string
           value: string
           effect: string
           timeAdded: time
      replicas: int32
      vmClass: string
      storageClass: string
      volumes:
        - name: string
          mountPath: string
          capacity:
            storage: size in GiB
```

```
      tkr:
        reference:
          name: string
      nodeDrainTimeout: string
  settings:
    storage:
      classes: [string]
      defaultClass: string
    network:
      cni:
        name: string
      pods:
        cidrBlocks: [string]
      services:
        cidrBlocks: [string]
      serviceDomain: string
      proxy:
        httpProxy: string
        httpsProxy: string
        noProxy: [string]
      trust:
        additionalTrustedCAs:
          - name: string
            data: string
```

Mit Anmerkungen versehene v1alpha3-API des TanzuKubernetesCluster

Die mit Anmerkungen versehene Spezifikation listet alle verfügbaren Parameter für die Bereitstellung eines TanzuKubernetesClusters mithilfe der v1alpha3-API mit Dokumentation für jedes Feld auf.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
#valid config key must consist of alphanumeric characters, '-', '_' or '.'
#metadata defines cluster information
metadata:
  #name for this Tanzu Kubernetes cluster
  name: string
  #namespace vSphere Namespace where to provision this cluster
  namespace: string
  #Use annotation to provision non-default OS for the VM nodes
  #PhotonOS is the default OS; use "ubuntu" to specify Ubuntu TKR
  annotations:
    run.tanzu.vmware.com/resolve-os-image: os-name=string
#spec defines cluster configuration
spec:
  #topology describes the number, purpose, organization of nodes
  #and the resources allocated for each
  #nodes are grouped into pools based on their purpose
  #controlPlane is special kind of a node pool
  #nodePools is for groups of worker nodes
  #each node pool is homogeneous: its nodes have the same
  #resource allocation and use the same storage
  topology:
```

```
    #controlPlane defines the topology of the cluster
    #controller, including the number of nodes and
    #the resources allocated for each
    #control plane must have an odd number of nodes
    controlPlane:
      #replicas is the number of nodes in the pool
      #the control plane can have 1 or 3 nodes
      #NOTE: production deployments require 3 control plane nodes
      #defaults to 1 if nil (empty)
      replicas: int32
      #vmClass is the name of the VirtualMachineClass
      #which describes the virtual hardware settings
      #to be used for each node in the node pool
      #vmClass controls the CPU and memory available
      #to the node and the requests and limits on
      #those resources; to list available vm classes run
      #kubectl get virtualmachineclass
      vmClass: string
      #storageClass to be used for storage of the disks
      #which store the root filesystems of the nodes
      #to list available storage classes run
      #kubectl describe storageclasses
      storageClass: string
      #volumes is the optional set of PVCs
      #to create and attach to each control plane node
      volumes:
        #name of the PVC to be used as the suffix (node.name)
        - name: string
          #mountPath is the directory where the volume
          #device is mounted; takes the form /dir/path
          mountPath: string
          #capacity is the PVC capacity
          capacity:
            #storage to be used for the disk
            #volume; if not specified defaults to
            #spec.controlPlane.storageClass
            storage: size in GiB
      #tkr.reference.name is the TKR NAME
      #to be used by control plane nodes
      #format is v1.27.11---vmware.1-fips.1-tkg.2
      #currently all tkr.reference.name fields must match
      tkr:
        reference:
          name: string
      #nodeDrainTimeout is the total amount of time
      #the controller will spend draining a node
      #the default value is 0 which means the node is
      #drained without any time limit
      nodeDrainTimeout: string
    #nodePools is an array that describes a group of
    #worker nodes in the cluster with the same configuration
    nodePools:
    #name of the worker node pool
    #must be unique in the cluster
    - name: string
```

```
      #failureDomain is the name of a vSphere Zone
      #failureDomain is required for multi-zoned Supervisor
      #in a multi-zoned Supervisor, you will have 3 node pools
      #each refernecing a different failureDomain zone name
      #refer to the examples
      failureDomain: string
      #labels are an optional map of string keys and values
      #to organize and categorize objects
      #propagated to the created nodes
      labels: map[string]string
      #taints specifies optional taints to register the
      #Node API object with; user-defined taints are
      #propagated to the created nodes
      taints:
        #key is the taint key to be applied to a node
        -  key: string
        #value is the taint value corresponding to the key
           value: string
        #effect is the effect of the taint on pods
        #that do not tolerate the taint; valid effects are
        #NoSchedule, PreferNoSchedule, NoExecute
           effect: string
        #timeAdded is the time when the taint was added
        #only written by the system for NoExecute taints
           timeAdded: time
      #replicas is the number of nodes in the pool
      #worker nodePool can have from 0 to 150 nodes
      #value of nil means the field is not reconciled,
      #allowing external services like autoscalers
      #to choose the number of nodes for the nodePool
      #by default CAPI's MachineDeployment will pick 1
      #NOTE: a cluster provisioned with 0 worker nodes/nodepools
      #is not assigned any load balancer services
      replicas: int32
      #vmClass is the name of the VirtualMachineClass
      #which describes the virtual hardware settings
      #to be used for each node in the pool
      #vmClass controls the CPU and memory available
      #to the node and the requests and limits on
      #those resources; to list available vm classes run
      #kubectl get virtualmachineclass
      vmClass: string
      #storageClass to be used for storage of the disks
      #which store the root filesystems of the nodes
      #to list available storage classes run
      #kubectl describe ns
      storageClass: string
      #volumes is the optional set of PVCs to create
      #and attach to each node for high-churn worker node
      #components such as the container runtime
      volumes:
        #name of this PVC to be used as the suffix (node.name)
        - name: string
          #mountPath is the directory where the volume
          #device is mounted; takes the form /dir/path
```

```
          mountPath: string
          #capacity is the PVC capacity
          capacity:
            #storage to be used for the disk
            #volume; if not specified defaults to
            #topology.nodePools[*].storageClass
            storage: size in GiB
      #tkr.reference.name points to the TKR NAME
      #to be used by spec.topology.nodePools[*] nodes
      #format is v1.27.11---vmware.1-fips.1-tkg.2
      #currently all tkr.reference.name fields must match
      tkr:
        reference:
          name: string
      #nodeDrainTimeout is the total amount of time
      #the controller will spend draining a node
      #the default value is 0 which means the node is
      #drained without any time limit
      nodeDrainTimeout: string
  #settings are optional runtime configurations
  #for the cluster, including persistent storage
  #for pods and node network customizations
  settings:
    #storage defines persistent volume (PV) storage entries
    #for container workloads; note that the storage used for
    #node disks is defined by topology.controlPlane.storageClass
    #and by spec.topology.nodePools[*].storageClass
    storage:
      #classes is a list of persistent volume (PV) storage
      #classes to expose for container workloads on the cluster
      #any class specified must be associated with the
      #vSphere Namespace where the cluster is provisioned
      #if omitted, all storage classes associated with the
      #namespace will be exposed in the cluster
      classes: [string]
      #defaultClass treats the named storage class as the default
      #for the cluster; because all namespaced storage classes
      #are exposed if specific classes are not named,
      #classes is not required to specify a defaultClass
      #many workloads, including TKG Extensions and Helm,
      #require a default storage class
      #if omitted, no default storage class is set
      defaultClass: string
    #netowrk defines custom networking for cluster workloads
    network:
      #cni identifies the CNI plugin for the cluster
      #use to override the default CNI set in the
      #tkgservicesonfiguration spec, or when customizing
      #network settings for the default CNI
      cni:
        #name is the name of the CNI plugin to use
        #supported values are antrea, calico, antrea-nsx-routed
        name: string
      #pods configures custom networks for pods
      #defaults to 192.168.0.0/16 if CNI is antrea or calico
```

```
    #defaults to empty if CNI is antrea-nsx-routed
    #custom subnet size must equal or exceed /24
    #use caution before seeting CIDR range other than /16
    #cannot overlap with Supervisor workload network
    pods:
      #cidrBlocks is an array of network ranges
      #multiple ranges may not be supported by all CNI plugins
      cidrBlocks: [string]
    #services configures custom network for services
    #defaults to 10.96.0.0/12
    #cannot overlap with Supervisor workload network
    services:
      #cidrBlocks is an array of network ranges
      #multiple ranges many not be supported by all CNI plugins
      cidrBlocks: [string]
    #serviceDomain specifies the service domain for the cluster
    #defaults to cluster.local
    serviceDomain: string
    #proxy configures proxy server to be used inside the cluster
    #if omitted no proxy is configured
    proxy:
      #httpProxy is the proxy URI for HTTP connections
      #to endpoints outside the cluster
      #takes form http://<user>:<pwd>@<ip>:<port>
      httpProxy: string
      #httpsProxy is the proxy URL for HTTPS connections
      #to endpoints outside the cluster
      #takes the frorm http://<user>:<pwd>@<ip>:<port>
      httpsProxy: string
      #noProxy is the list of destination domain names, domains,
      #IP addresses, and other network CIDRs to exclude from proxying
      #must include Supervisor Cluster Pod, Egress, Ingress CIDRs
      noProxy: [string]
    #trust configures additional certificates for the cluster
    #if omitted no additional certificate is configured
    trust:
      #additionalTrustedCAs are additional trusted certificates
      #can be additional CAs or end certificates
      additionalTrustedCAs:
        #name is the name of the additional trusted certificate
        #must match the name used in the filename
        - name: string
          #data holds the contents of the additional trusted cert
          #PEM Public Certificate data as a base64-encoded string
          #such as LS0tLS1C...LS0tCg== where "..." is the
          #middle section of the long base64-encoded string
          data: string
```

v1alpha3-Beispiel: Standard-TanzuKubernetesCluster

In der Beispiel-YAML finden Sie Informationen zur Bereitstellung eines Standard-Tanzu Kubernetes-Clusters (TanzuKubernetesCluster) mithilfe der v1alpha3-API.

v1alpha3-Beispiel: Standard-TanzuKubernetesCluster

Mit der folgenden Beispiel-YAML wird ein Standard-Tanzu Kubernetes-Cluster mithilfe der v1alpha3-API bereitgestellt.

Dieses Beispiel zeigt die für die Bereitstellung eines TKC erforderliche Mindestkonfiguration. Da die standardmäßigen Netzwerk- und Speichereinstellungen verwendet werden, werden sie aus der YAML ausgeschlossen. Der referenzierte TKR wird sowohl für die Steuerungsebene als auch für die Worker-Knoten verwendet.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-default
  namespace: tkg-cluster-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
    nodePools:
    - replicas: 3
      name: worker
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
```

v1alpha3-Beispiel: TKC mit Standardspeicher und Knoten-Volumes

In der Beispiel-YAML finden Sie Informationen dazu, wie Sie einen Tanzu Kubernetes-Cluster (TanzuKubernetesCluster) mithilfe der v1alpha3-API mit einer Standardspeicherklasse und benutzerdefinierten Einstellungen für Knoten-Volumes bereitstellen.

v1alpha3-Beispiel: TKC mit Knoten-Volumes und Standardspeicher

Mit der folgenden Beispiel-YAML wird ein benutzerdefinierter Tanzu Kubernetes-Cluster mithilfe der v1alpha3-API bereitgestellt.

Beachten Sie die folgenden optionalen Anpassungen in diesem Beispiel. Weitere Informationen finden Sie in der TanzuKubernetesCluster v1alpha3-API.

- Der Cluster wird mit einer Standardspeicherklasse bereitgestellt, die für einige Tools erforderlich ist, beispielsweise für von Helm- und Tanzu-Paketen bereitgestellte Arbeitslasten.

- Worker-Knoten-Volumes werden für Komponenten mit hoher Änderungsrate deklariert, wie z. B. `containerd` und `kubelet`

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-custom-storage
  namespace: tkg-cluster-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
    nodePools:
    - replicas: 3
      name: worker-np
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
      volumes:
      - name: containerd
        mountPath: /var/lib/containerd
        capacity:
          storage: 50Gi
      - name: kubelet
        mountPath: /var/lib/kubelet
        capacity:
          storage: 50Gi
  settings:
    storage:
      defaultClass: tkg-storage-policy
```

v1alpha3-Beispiel: TKC mit benutzerdefiniertem Netzwerk

In der Beispiel-YAML finden Sie Informationen dazu, wie Sie einen Tanzu Kubernetes-Cluster (TanzuKubernetesCluster) mithilfe der v1alpha3-API mit benutzerdefinierten Netzwerkeinstellungen bereitstellen.

v1alpha3-Beispiel: TKC mit benutzerdefinierten Netzwerkeinstellungen

Das Netzwerk wird wie folgt angepasst. Weitere Informationen finden Sie in der TanzuKubernetesCluster v1alpha3-API.

- Anstelle der standardmäßigen Antrea-CNI wird die Calico-CNI verwendet.
- Es werden nicht standardmäßige Subnetze für Pods und Dienste verwendet.

- Ein Proxyserver und TLS-Zertifikate werden deklariert

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-custom-network
  namespace: tkg2-cluster-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
    nodePools:
    - name: worker
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
      volumes:
      - name: containerd
        mountPath: /var/lib/containerd
        capacity:
          storage: 50Gi
      - name: kubelet
        mountPath: /var/lib/kubelet
        capacity:
          storage: 50Gi
  settings:
    storage:
      defaultClass: tkg-storage-policy
    network:
      cni:
        name: calico
      services:
        cidrBlocks: ["172.16.0.0/16"]
      pods:
        cidrBlocks: ["192.168.0.0/16"]
      serviceDomain: cluster.local
      proxy:
        httpProxy: http://<user>:<pwd>@<ip>:<port>
        httpsProxy: http://<user>:<pwd>@<ip>:<port>
        noProxy: [10.246.0.0/16,192.168.144.0/20,192.168.128.0/20]
      trust:
        additionalTrustedCAs:
          - name: CompanyInternalCA-1
            data: LS0tLS1C...LS0tCg==
          - name: CompanyInternalCA-2
            data: MTLtMT1C...MT0tPg==
```

Überlegungen zum Anpassen des TKC-Pod-Netzwerks

Die Einstellung `spec.settings.network.pods.cidrBlocks` der Clusterspezifikation ist standardmäßig auf 192.168.0.0/16 festgelegt.

Im Fall einer Anpassung liegt die minimale CIDR-Blockgröße der Pods bei /24. Seien Sie jedoch vorsichtig, wenn Sie die `pods.cidrBlocks`-Subnetzmaske über /16 hinaus einschränken.

TKG weist jedem Clusterknoten ein /24-Subnetz zu, das aus `pods.cidrBlocks` abgeteilt wird. Diese Zuteilung wird vom Kubernetes Controller Manager > NodeIPAMController-Parameter mit der Bezeichnung NodeCIDRMaskSize bestimmt, der die Größe der Subnetzmaske für das Knoten-CIDR im Cluster festlegt. Die Subnetzmaske für den Standardknoten beträgt /24 für IPv4.

Da jeder Knoten in einem Cluster ein /24-Subnetz aus `pods.cidrBlocks` erhält, kann es zu einem Mangel an IP-Adressen kommen, wenn Sie eine Subnetzmaskengröße verwenden, die für den bereitzustellenden Cluster zu restriktiv ist.

Die folgenden Knotengrenzwerte gelten für einen Tanzu Kubernetes-Cluster, der entweder mit der Antrea- oder der Calico-CNI bereitgestellt wird.

/16 == max. 150 Knoten (pro ConfigMax)

/17 == max. 128 Knoten

/18 == max. 64 Knoten

/19 == max. 32 Knoten

/20 == max. 16 Knoten

/21 == max. 8 Knoten

/22 == max. 4 Knoten

/23 == max. 2 Knoten

/24 == max. 1 Knoten

v1alpha3-Beispiel: TKC mit Ubuntu-TKR

In der Beispiel-YAML finden Sie Informationen zur Bereitstellung eines TanzuKubernetesCluster-Clusters, der das Ubuntu-Betriebssystem für Clusterknoten verwendet. Ein solcher Cluster kann für vGPU-Arbeitslasten verwendet werden.

v1alpha3-Beispiel: TKC mit Ubuntu-TKR

Standardmäßig wird die Photon OS-Edition der benannten TKR für TKG-Clusterknoten verwendet. Wenn die referenzierte TKR das OSImage-Format unterstützt und eine Ubuntu-Betriebssystemedition verfügbar ist, geben Sie mithilfe der Anmerkung `run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu` die Ubuntu-Betriebssystemedition der TKR an. Weitere Informationen zum OSImage-Format finden Sie unter TKr-Betriebssystem-Image-Format.

Für KI-/ML-Arbeitslasten ist die Ubuntu-TKR erforderlich. Jeder Worker-Knotenpool verfügt über ein separates Volume für die containerd-Laufzeit und kubelet mit jeweils 70 GiB Kapazität. Es wird empfohlen, ein separates Volume mit dieser Größe für containerbasierte KI-/ML-Arbeitslasten bereitzustellen.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-ubuntu-gpu
  namespace: tkg-cluster-ns
  annotations:
    run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
spec:
   topology:
     controlPlane:
       replicas: 3
       storageClass: tkg-storage-policy
       vmClass: guaranteed-large
       tkr:
         reference:
           name: v1.25.7---vmware.3-fips.1-tkg.1
     nodePools:
     - name: nodepool-a100-primary
       replicas: 3
       storageClass: tkg-storage-policy
       vmClass: vgpu-a100
       tkr:
         reference:
           name: v1.25.7---vmware.3-fips.1-tkg.1
       volumes:
       - name: containerd
         mountPath: /var/lib/containerd
         capacity:
           storage: 70Gi
       - name: kubelet
         mountPath: /var/lib/kubelet
         capacity:
           storage: 70Gi
     - name: nodepool-a100-secondary
       replicas: 3
       storageClass: tkg-storage-policy
       vmClass: vgpu-a100
       tkr:
         reference:
           name: v1.25.7---vmware.3-fips.1-tkg.1
       volumes:
       - name: containerd
         mountPath: /var/lib/containerd
         capacity:
           storage: 70Gi
       - name: kubelet
         mountPath: /var/lib/kubelet
         capacity:
           storage: 70Gi
```

```
    settings:
      storage:
        defaultClass: tkg-storage-policy
      network:
        cni:
         name: antrea
        services:
         cidrBlocks: ["198.51.100.0/12"]
        pods:
         cidrBlocks: ["192.0.2.0/16"]
        serviceDomain: cluster.local
```

v1alpha3-Beispiel: TKC in verschiedenen vSphere-Zonen

In der Beispiel-YAML finden Sie Informationen zur Bereitstellung eines Tanzu Kubernetes-Clusters (TanzuKubernetesCluster) in verschiedenen vSphere-Zonen mithilfe der v1alpha3-API.

vSphere-Zonen und Fehlerdomänen

vSphere-Zonen bieten die Möglichkeit, hochverfügbare TKG-Cluster auf Supervisor zu erstellen. Wenn Sie einen TKG-Cluster in verschiedenen vSphere-Zonen bereitstellen, müssen Sie die Fehlerdomäne für jeden Knotenpool angeben.

Jede Fehlerdomäne wird einer vSphere-Zone zugeordnet, die dadurch mit einem vSphere-Cluster verknüpft wird. vSphere-Fehlerdomänen werden vom vSphere-Administrator beim Erstellen von vSphere-Zonen definiert und verwaltet. Das für den TKG-Cluster verwendete Speicherprofil muss als `zonal` konfiguriert sein. Weitere Informationen finden Sie unter Erstellen einer vSphere-Speicherrichtlinie für TKG-Dienst-Cluster.

Wenn Sie Pods mit Replikaten in einem TKG-Cluster auf Supervisor bereitstellen, werden die Pod-Instanzen automatisch auf die vSphere-Zonen verteilt. Sie müssen bei der POD-Bereitstellung auf dem TKG-Cluster keine Zonendetails angeben.

Um die Verfügbarkeit von vSphere-Zonen in der TKG-Umgebung zu überprüfen, führen Sie einen der folgenden Befehle in dem vSphere-Namespace aus, in dem Sie den TKG-Cluster bereitstellen:

```
kubectl get vspherezones
```

```
kubectl get availabilityzones
```

Beide Befehle stehen `system:authenticated`-Benutzern zur Verfügung. vSphere-Zonen sind Ressourcen mit Supervisor-Geltungsbereich, daher müssen Sie keinen Namespace angeben.

v1alpha3-Beispiel: TKC in verschiedenen vSphere-Zonen

Die Beispiel-YAML stellt einen TKG-Cluster in verschiedenen vSphere-Zonen bereit.

In diesem Beispiel geben Sie die vSphere-Zone im Parameter „`failureDomain`“ für jeden nodePool an. Der Wert des Parameters ist der Name der vSphere-Zone.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-zoned
  namespace: tkg-cluster-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: tkg2-storage-policy-zonal
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
    nodePools:
      - name: nodepool-a01
        replicas: 3
        vmClass: guaranteed-medium
        storageClass: tkg-storage-policy-zonal
        failureDomain: az1
      - name: nodepool-a02
        replicas: 3
        vmClass: guaranteed-medium
        storageClass: tkg-storage-policy-zonal
        failureDomain: az2
      - name: nodepool-a03
        replicas: 3
        vmClass: guaranteed-medium
        storageClass: tkg-storage-policy-zonal
        failureDomain: az3
  settings:
    storage:
      defaultClass: tkg-storage-policy-zonal
    network:
      cni:
        name: antrea
      services:
        cidrBlocks: ["198.51.100.0/12"]
      pods:
        cidrBlocks: ["192.0.2.0/16"]
      serviceDomain: cluster.local
```

v1alpha3-Beispiel: TKC mit routingfähigem Pod-Netzwerk

Sie können einen Tanzu Kubernetes-Cluster (TanzuKubernetesCluster) mit routingfähigen Pods-Netzwerken erstellen, indem Sie ein routingfähiges Namespace-Netzwerk auf Supervisor konfigurieren und `antrea-nsx-routed` als CNI für den Cluster angeben.

Informationen zum Netzwerk von routingfähigen Pods

Wenn Sie einen Tanzu Kubernetes-Cluster mithilfe der Plug-Ins `antrea` oder `calico` bereitstellen, erstellt das System das Standardnetzwerk für Pods `192.168.0.0/16`. Dieses Subnetz ist ein privater Adressbereich, der nur innerhalb des Clusters eindeutig und nicht im Netzwerk routingfähig ist.

Die TKG-v1alpha3-API unterstützt routingfähige Pod-Netzwerke mithilfe des `antrea-nsx-routed`-CNI-Plug-Ins. Diese Netzwerkschnittstelle ist ein angepasstes Antrea-Plug-In, das zur Unterstützung routingfähiger Pod-Netzwerke für TKG-Cluster konfiguriert ist. In der Clusterspezifikation muss das Feld „pods CIDR blocks“ explizit Null sein, damit die IP-Adressenverwaltung (IP Address Management, IPAM) vom Supervisor verarbeitet wird. Weitere Informationen finden Sie in dem folgenden Beispiel.

Durch die Aktivierung eines routingfähigen Pod-Netzwerks können Pods direkt von einem Client adressiert werden, der sich außerhalb des Clusters befindet. Darüber hinaus werden Pod-IP-Adressen beibehalten, sodass externe Netzwerkdienste und -server die Quell-Pods identifizieren und Richtlinien basierend auf IP-Adressen anwenden können. Unterstützte Datenverkehrsmuster, einschließlich der folgenden:

- Der Datenverkehr ist zwischen einem TKG-Cluster-Pod und einem vSphere Pod in demselben vSphere-Namespace zulässig.
- Der Datenverkehr wird zwischen einem TKG-Cluster-Pod und einem vSphere Pod in demselben vSphere-Namespaces gelöscht.
- Knoten der Supervisor-Steuerungsebene können TKG-Cluster-Pods erreichen.
- TKG-Cluster-Pods können das externe Netzwerk erreichen.
- Externes Netzwerk kann TKG-Cluster-Pods nicht erreichen. Der Datenverkehr wird von den Isolierungsregeln der verteilten Firewall (DFW) auf den Clusterknoten verworfen.

Erstellen eines routingfähigen Pod-Netzwerks: Supervisor-Konfiguration

Zum Erstellen eines routingfähigen Pod-Netzwerks ist eine Konfiguration auf dem Supervisor und im TKG-Cluster erforderlich.

Hinweis Supervisor muss mit NSX konfiguriert werden, um routingfähige Pods-Netzwerke zu verwenden. Sie können keine routingfähigen Pods mit dem VDS-Netzwerk verwenden.

So konfigurieren Sie ein routingfähiges Pod-Netzwerk auf Supervisor:

1 Erstellen Sie einen neuen vSphere-Namespace.

   Weitere Informationen finden Sie unter Erstellen eines vSphere-Namespace für das Hosting von TKG-Dienst-Clustern.

2 Aktivieren Sie das Kontrollkästchen **Supervisor-Netzwerkeinstellungen außer Kraft setzen**.

   Weitere Informationen finden Sie unter Überschreiben der Einstellungen für das Arbeitslastennetzwerk für einen vSphere-Namespace.

3 Konfigurieren Sie das routingfähige Pod-Netzwerk wie folgt.

Bereich	Beschreibung
NAT-Modus	Da Sie ein routingfähiges Subnetz verwenden, deaktivieren Sie diese Option, um die Netzwerkadressübersetzung (Network Address Translation, NAT) zu deaktivieren.
Namespace-Netzwerk-CIDR	Das Namespace-Netzwerk-CIDR ist ein Subnetz, das als IP-Pool für den vSphere-Namespace betrieben wird. Mit dem Namespace-Subnetzpräfix wird die Größe eines beliebigen nachfolgenden CIDR-Blocks beschrieben, der aus diesem IP-Pool entfernt wird. Befüllen Sie dieses Feld mit einem routingfähigen IP-Subnetz im Format „IP-Adresse/Bit“ (z. B. 10.0.0.6/16). NCP erstellt einen oder mehrere IP-Pools aus den für das Netzwerk angegebenen IP-Blöcken. Sie sollten mindestens eine /23-Subnetzgröße angeben. Wenn Sie beispielsweise ein routingfähiges /23-Subnetz mit einem /28-Subnetzpräfix angeben, erhalten Sie 32 Subnetze, die für einen Cluster mit 6 Knoten ausreichen sollten. Ein /24-Subnetz mit einem /28-Präfix ergibt nur 2 Subnetze, die nicht ausreichen.
Namespace-Subnetzpräfix	Mit dem Namespace-Subnetzpräfix wird die Größe eines beliebigen nachfolgenden CIDR-Blocks beschrieben, der aus diesem IP-Pool des Namespace-Netzwerks entfernt wird. Geben Sie beispielsweise ein Subnetzpräfix im Format „/28“ an.

4 Klicken Sie auf **Erstellen**, um ein Netzwerk routingfähiger Pods zu erstellen.

Erstellen eines routingfähigen Pod-Netzwerks: TKG-Clusterkonfiguration

Die folgende Beispiel-YAML zeigt, wie ein Cluster mit einem Netzwerk routingfähiger Pods konfiguriert wird.

Die Clusterspezifikation deklariert `antrea-nsx-routed` als CNI, um Netzwerke routingfähiger Pods zu aktivieren. Wenn `antrea-nsx-routed` angegeben ist, schlägt die Clusterbereitstellung fehl, wenn kein NSX-T-Netzwerk verwendet wird.

Wenn die CNI als „`antrea-nsx-routed`“ angegeben ist, muss das Feld „`pods.cidrBlock`“ leer sein.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-routable-pods
  namespace: tkg-cluster-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
```

```
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
    nodePools:
    - name: worker-nodepool-a1
      replicas: 3
      vmClass: guaranteed-large
      storageClass: tkg-storage-policy
      tkr:
        reference:
          name: v1.25.7---vmware.3-fips.1-tkg.1
  settings:
    storage:
      defaultClass: tkg-storage-policy
    network:
      #antrea-nsx-routed is the required CNI
      cni:
        name: antrea-nsx-routed
      services:
        cidrBlocks: ["10.97.0.0/24"]
      #pods.cidrBlocks must be null (empty)
      pods:
        cidrBlocks:
      serviceDomain: cluster.local
```

v1alpha3-Beispiel: TKC mit zusätzlichen vertrauenswürdigen CA-Zertifikaten für SSL/TLS

In der Beispiel-YAML finden Sie Informationen zur Bereitstellung eines TanzuKubernetesClusters unter Verwendung der v1alpha3-API mit zusätzlichen vertrauenswürdigen CA-Zertifikaten für SSL/TLS.

v1alpha3-Beispiel: TKC mit zusätzlichen vertrauenswürdigen CA-Zertifikaten

Der Cluster wird folgendermaßen angepasst. Weitere Informationen finden Sie in der TanzuKubernetesCluster v1alpha3-API.

- Zusätzliche vertrauenswürdige CA-Zertifikate werden im Abschnitt `network.trust.additionalTrustedCAs` der Clusterspezifikation deklariert.
- Das Feld `additionalTrustedCAs` stellt ein Array von Name/Wert-Paaren dar:
  - Das Feld `name` ist eine benutzerdefinierte Zeichenfolge.
  - Bei dem Wert `data` handelt es sich um den Inhalt des CA-Zertifikats im PEM-Format, das Base64-codiert ist.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc-additional-trusted-cas
  namespace: tkgs-cluster-ns
spec:
```

```
topology:
  controlPlane:
    replicas: 3
    vmClass: guaranteed-medium
    storageClass: tkgs-storage-policy
    tkr:
      reference:
        name: v1.25.7---vmware.3-fips.1-tkg.1
  nodePools:
  - name: worker
    replicas: 3
    vmClass: guaranteed-medium
    storageClass: tkgs-storage-policy
    tkr:
      reference:
        name: v1.25.7---vmware.3-fips.1-tkg.1
settings:
  storage:
    defaultClass: tkgs-storage-policy
  network:
    trust:
      additionalTrustedCAs:
        - name: CompanyInternalCA-1
          data: LS0tLS1C...LS0tCg==
        - name: CompanyInternalCA-2
          data: MTLtMT1C...MT0tPg==
```

Verfahren: Neuer Cluster

Führen Sie folgendes Verfahren aus, um mindestens ein zusätzliches vertrauenswürdiges CA-Zertifikat in einen neuen TKGS-Cluster aufzunehmen.

1 Befüllen Sie das Feld `additionalTrustedCAs` mit dem Namen und dem Datenwert für ein oder mehrere CA-Zertifikate.

2 Stellen Sie den Cluster wie gewohnt bereit.

Weitere Informationen hierzu finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl.

3 Nach der erfolgreichen Bereitstellung des Clusters werden die von Ihnen hinzugefügten CA-Zertifikate vom Cluster als vertrauenswürdig eingestuft.

Verfahren: Vorhandener Cluster

Führen Sie das folgende Verfahren aus, um einem vorhandenen Cluster mindestens ein zusätzliches vertrauenswürdiges CA-Zertifikat hinzuzufügen.

1 Stellen Sie sicher, dass Sie die kubectl-Bearbeitung konfiguriert haben.

Weitere Informationen hierzu finden Sie unter Konfigurieren eines Texteditors für Kubectl.

2 Bearbeiten Sie die Clusterspezifikation.

```
kubectl edit tanzukubernetescluster/tkgs-cluster-name
```

3 Fügen Sie den Abschnitt `network.trust.additionalTrustedCAs` zur Spezifikation hinzu.

4 Befüllen Sie das Feld `additionalTrustedCAs` mit dem Namen und dem Datenwert für ein oder mehrere CA-Zertifikate.

5 Speichern Sie die Änderungen im Texteditor und stellen Sie sicher, dass die Änderungen von kubectl registriert wurden.

```
kubectl edit tanzukubernetescluster/tkgs-cluster-name
tanzukubernetescluster.run.tanzu.vmware.com/tkgs-cluster-name edited
```

6 Wenn ein paralleles Update für den Cluster initiiert wird, werden die zusätzlichen vertrauenswürdigen CA-Zertifikate hinzugefügt.

Weitere Informationen hierzu finden Sie unter Grundlegendes zum Modell für parallele Updates für TKG-Dienstcluster.

Überprüfen zusätzlicher vertrauenswürdiger CA-Zertifikate

Die zusätzlichen vertrauenswürdigen CA-Zertifikate, die dem Cluster hinzugefügt wurden, sind in der Kubeconfig-Datei für den Cluster enthalten.

Fehlerbehebung bei zusätzlichen vertrauenswürdigen CA-Zertifikaten

Weitere Informationen hierzu finden Sie unter Fehlerbehebung bei zusätzlichen vertrauenswürdigen Zertifizierungsstellen.

Anwendungsbeispiele

Der häufigste Anwendungsfall besteht im Hinzufügen einer zusätzlichen vertrauenswürdigen Zertifizierungsstelle zum Herstellen einer Verbindung mit einer Containerregistrierung. Weitere Informationen hierzu finden Sie unter Integrieren von TKG-Dienst-Clustern in eine private Containerregistrierung.

Betreiben von TKG-Dienstclustern

8

Dieser Abschnitt enthält Themen zum Betrieb von TKG-Dienstclustern.

Lesen Sie als Nächstes die folgenden Themen:

- Konfigurieren eines Texteditors für Kubectl
- Manuelles Skalieren eines Clusters mithilfe von „Kubectl“
- Überwachen des TKG-Clusterstatus mithilfe des vSphere Client
- Überwachen des TKG-Clusterstatus mithilfe von kubectl
- Überprüfen der TKG-Clusterbereitschaft mithilfe von Kubectl
- Überprüfen der Integrität der TKG-Clustermaschine mithilfe von Kubectl
- Überprüfen der Integrität des TKG-Clusters mithilfe von Kubectl
- Überprüfen der Integrität des TKG-Cluster-Volumes mithilfe von Kubectl
- Überwachen der Volume-Integrität in einem Tanzu Kubernetes Grid-Cluster
- Überwachen von dauerhaften Volumes mithilfe des vSphere Client
- Abrufen von geheimen Schlüsseln für TKG-Cluster mithilfe von Kubectl
- Überprüfen des TKG-Clusternetzwerks mithilfe von Kubectl
- Überprüfen von TKG-Clustervorgängen mithilfe von Kubectl
- Anzeigen des Lebenszyklusstatus von TKG-Clustern
- Anzeigen der Ressourcenhierarchie für einen TKG-Cluster mithilfe von kubectl
- Konfigurieren von MachineHealthCheck für v1beta1-Cluster

Konfigurieren eines Texteditors für Kubectl

Für den Betrieb und die Pflege von TKG-Clustern konfigurieren Sie einen Standard-Texteditor für kubectl.

Verwenden des Befehls „kubectl edit“

Nachdem Sie einen TKG-Cluster bereitgestellt haben, müssen Sie ihn betreiben und pflegen. Typische Aufgaben sind das Skalieren von Clusterknoten und das Aktualisieren der TKR-Version. Zur Durchführung derartiger Aufgaben aktualisieren Sie das Cluster-Manifest mit dem Befehl kubectl edit.

Mit dem Befehl `kubectl edit CLUSTER-KIND/CLUSTER-NAME` wird das Cluster-Manifest in dem durch die Umgebungsvariable KUBE_EDITOR oder EDITOR definierten Texteditor geöffnet. Wenn Sie die Manifeständerungen speichern, meldet `kubectl`, dass die Bearbeitungen erfolgreich aufgezeichnet wurden, und der Cluster wird mit den Änderungen aktualisiert.

Beispiel:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkg-cluster-1 edited
```

Um Änderungen zu verwerfen, schließen Sie den Editor, ohne zu speichern.

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
Edit cancelled, no changes made.
```

Konfigurieren des Befehls „kubectl edit“

Um den Befehl `kubectl edit` zu verwenden, wird in Linux die Umgebungsvariable EDITOR festgelegt. Erstellen Sie andernfalls die Umgebungsvariable KUBE_EDITOR und geben Sie Ihren bevorzugten Texteditor als Variablenwert an. Hängen Sie das Watch-Flag (`-w`) an, damit kubectl weiß, wann Sie Ihre Änderungen bestätigt (gespeichert) haben.

Beachten Sie die spezifischen Anweisungen für Ihr Betriebssystem.

Linux

Um `kubectl edit` unter Linux (z. B. Ubuntu) zu konfigurieren, ist Vim der Standardbefehlszeilen-`EDITOR`. Wenn dies zutrifft, ist keine weitere Aktion erforderlich, um den Befehl `kubectl edit` zu verwenden.

Wenn Sie einen anderen Texteditor verwenden möchten, erstellen Sie eine Umgebungsvariable namens `KUBE_EDITOR`, deren Wert auf den Pfad des bevorzugten Texteditors festgelegt ist.

Mac OS

Um `kubectl edit` unter Mac OS zu konfigurieren, erstellen Sie eine Umgebungsvariable namens `KUBE_EDITOR`, deren Wert auf den Pfad Ihres bevorzugten Texteditors festgelegt ist. Hängen Sie das Wait-Flag (`--wait` oder die Tastenkombination `-w`) an den Wert an, damit der Editor weiß, wann Sie Ihre Änderungen bestätigt (gespeichert) haben.

Beispielsweise legt der folgende Zusatz zum `.bash_profile` Sublime als Standardtexteditor für kubectl fest und enthält das Wait-Flag. Dadurch weiß der Editor, wann Sie Änderungen gespeichert haben.

```
export KUBE_EDITOR="/Applications/Sublime.app/Contents/SharedSupport/bin/subl -w"
```

Windows

Um `kubectl edit` unter Windows zu konfigurieren, erstellen Sie eine Systemumgebungsvariable namens `KUBE_EDITOR`, deren Wert auf den Pfad Ihres bevorzugten Texteditors festgelegt ist. Hängen Sie das Watch-Flag (`-w`) an den Wert an.

Die folgende Umgebungsvariable legt beispielsweise Visual Studio Code als Standardtexteditor für kubectl fest und enthält das Watch-Flag, damit Kubernetes weiß, wann Sie Ihre Änderungen speichern:

```
KUBE_EDITOR=code -w
```

Um Sublime als kubectl-Editor unter Windows zu konfigurieren, fügen Sie das Sublime-Programmverzeichnis an den Systempfad an und erstellen eine Systemvariable für die ausführbare Sublime-Datei. Beispiel:

SYSTEMPFAD anhängen:

```
C:\Program Files\Sublime Text 3\
```

Name und Wert der Systemvariablen:

```
KUBE_EDITOR=sublime_text.exe -w
```

Manuelles Skalieren eines Clusters mithilfe von „Kubectl“

Sie können einen TKG-Dienstcluster durch Änderung der Knotenanzahl horizontal und durch Änderung der VM-Klasse zum Hosten der Knoten vertikal skalieren. Sie können auch Volumes skalieren, die an Clusterknoten angehängt sind.

Unterstützte manuelle Skalierungsvorgänge

In der Tabelle sind die unterstützten Skalierungsvorgänge für TKG-Cluster aufgeführt.

Tabelle 8-1. Unterstützte Skalierungsvorgänge für TKGS-Cluster

Knoten	Horizontales Hochskalieren	Horizontales Herunterskalieren	Vertikale Skalierung	Skalierung des Volumes
Steuerungsebene	Ja	Nein	Ja	Ja*
Worker	Ja	Ja	Ja	Ja

Berücksichtigen Sie die folgenden Aspekte:

- Die Anzahl der Steuerungsebenen-Knoten muss ungerade sein, entweder 1 oder 3. Die horizontale Skalierung der Steuerungsebene wird unterstützt, aber das vertikale Skalieren der Steuerungsebene wird nicht unterstützt. Weitere Informationen finden Sie unter Horizontales Hochskalieren der Steuerungsebene.
- Bei der vertikalen Skalierung eines Clusterknotens kann es vorkommen, dass Arbeitslasten mangels verfügbarer Ressourcen nicht mehr auf dem Knoten ausgeführt werden können. Aus diesem Grund wird horizontale Skalierung in der Regel bevorzugt. Weitere Informationen finden Sie unter Horizontales Hochskalieren der Worker-Knoten.
- VM-Klassen sind nicht unveränderlich. Wenn Sie einen TKG-Cluster nach der Bearbeitung einer von diesem Cluster verwendeten VM-Klasse skalieren, verwenden neue Clusterknoten die aktualisierte Klassendefinition, aber vorhandene Clusterknoten verwenden weiterhin die anfängliche Klassendefinition, was zu einer Nichtübereinstimmung führt. Weitere Informationen finden Sie unter Informationen zu VM-Klassen.
- Worker-Knoten-Volumes können nach der Bereitstellung geändert werden. Ebenso können Steuerungsebenenknoten geändert werden, sofern Sie vSphere 8 U3 oder höher und eine kompatible TKr verwenden. Weitere Informationen finden Sie unter Skalieren von Cluster-Knoten-Volumes.

Skalierungsvoraussetzungen: Konfigurieren der Kubectl-Bearbeitung

Um einen TKG-Cluster zu skalieren, aktualisieren Sie das Cluster-Manifest mit dem Befehl `kubectl edit CLUSTER-KIND/CLUSTER-NAME`. Wenn Sie die Manifest-Änderungen speichern, wird der Cluster mit den Änderungen aktualisiert. Weitere Informationen finden Sie unter Konfigurieren eines Texteditors für Kubectl.

Beispiel:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkg-cluster-1 edited
```

Um Änderungen zu verwerfen, schließen Sie den Editor, ohne zu speichern.

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
Edit cancelled, no changes made.
```

Horizontales Hochskalieren der Steuerungsebene

Skalieren Sie einen TKG-Cluster horizontal, indem Sie die Anzahl der Steuerungsebenenknoten von 1 auf 3 erhöhen.

Hinweis Produktionscluster benötigen 3 Knoten der Steuerungsebene.

1 Melden Sie sich bei Supervisor an.

```
kubectl vsphere login --server=SUPERVISOR-IP-ADDRESS --vsphere-username USERNAME
```

2 Wechseln Sie den Kontext zum vSphere-Namespace, in dem der TKG-Cluster ausgeführt wird.

```
kubectl config use-context tkg-cluster-ns
```

3 Listet die Kubernetes-Cluster auf, die im vSphere-Namespace ausgeführt werden.

Verwenden Sie die folgende Syntax:

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

Beispielsweise für einen v1alpha3-API-Cluster:

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

Beispielsweise für einen v1beta1-API-Cluster:

```
kubectl get cluster -n tkg-cluster-ns
```

4 Rufen Sie die Anzahl der im Zielcluster ausgeführten Knoten ab.

Für einen v1alpha3-API-Cluster:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

Der TKG-Cluster verfügt über 1 Steuerungsebenenknoten und 3 Worker-Knoten.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster-1    1               3        v1.24.9---vmware.1-tkg.4
5d12h   True
```

Für einen v1beta1-API-Cluster:

```
kubectl get cluster tkg-cluster-1
```

5 Führen Sie zum Laden des Cluster-Manifests zur Bearbeitung den Befehl `kubectl edit` aus.

Für einen v1alpha3-API-Cluster:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

Für einen v1beta1-API-Cluster:

```
kubectl edit cluster/tkg-cluster-1
```

Das Cluster-Manifest wird in dem Texteditor geöffnet, der von den Umgebungsvariablen KUBE_EDITOR oder EDITOR definiert wird.

6 Erhöhen Sie die Anzahl der Steuerungsebenenknoten im Abschnitt `spec.topology.controlPlane.replicas` des Manifests von 1 auf 3.

Für einen v1alpha3-API-Cluster:

```
...
spec:
  topology:
    controlPlane:
      replicas: 1
...
```

```
...
spec:
  topology:
    controlPlane:
      replicas: 3
...
```

Für einen v1beta1-API-Cluster:

```
...
spec:
  ...
  topology:
    class: tanzukubernetescluster
    controlPlane:
      metadata: {}
      replicas: 1
    variables:
...
```

```
...
spec:
  ...
  topology:
    class: tanzukubernetescluster
    controlPlane:
      metadata: {}
      replicas: 3
    variables:
...
```

7 Speichern Sie die Datei im Texteditor, um die Änderungen zu übernehmen. (Schließen Sie zum Abbrechen den Editor ohne Speichern.)

   Wenn Sie die Manifest-Änderungen speichern, wendet kubectl die Änderungen auf den Cluster an. Im Hintergrund stellt der VM-Dienst auf Supervisor den neuen Steuerungsebenenknoten bereit.

8 Vergewissern Sie sich, dass die neuen Knoten hinzugefügt wurden.

   Für einen v1alpha3-API-Cluster:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

Die horizontal hochskalierte Steuerungsebene verfügt jetzt über 3 Knoten.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster-1    3               3        v1.24.9---vmware.1-tkg.4
5d12h   True
```

Für einen v1beta1-API-Cluster:

```
kubectl get cluster tkg-cluster-1
```

Horizontales Hochskalieren der Worker-Knoten

Sie können einen TKG-Cluster horizontal skalieren, indem Sie die Zahl der Worker-Knoten erhöhen.

1 Melden Sie sich bei Supervisor an.

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

2 Wechseln Sie den Kontext zum vSphere-Namespace, in dem der TKG-Cluster ausgeführt wird.

```
kubectl config use-context tkg-cluster-ns
```

3 Listet die Kubernetes-Cluster auf, die im vSphere-Namespace ausgeführt werden.

Verwenden Sie die folgende Syntax:

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

Beispielsweise für einen v1alph3-API-Cluster:

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

Beispielsweise für einen v1beta1-API-Cluster:

```
kubectl get cluster -n tkg-cluster-ns
```

4 Rufen Sie die Anzahl der im Zielcluster ausgeführten Knoten ab.

Für einen v1alpha3-API-Cluster:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

Der folgende Cluster hat z. B. 3 Steuerungsebenenknoten und 3 Worker-Knoten.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster-1    3               3        v1.24.9---vmware.1-tkg.4
5d12h   True
```

Für einen v1beta1-API-Cluster:

```
kubectl get cluster tkg-cluster-1
```

5 Führen Sie zum Laden des Cluster-Manifests zur Bearbeitung den Befehl `kubectl edit` aus.

Für einen v1alpha3-API-Cluster:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

Für einen v1beta1-API-Cluster:

```
kubectl edit cluster/tkg-cluster-1
```

Das Cluster-Manifest wird in dem Texteditor geöffnet, der von den Umgebungsvariablen KUBE_EDITOR oder EDITOR definiert wird.

6 Erhöhen Sie die Anzahl der Worker-Knoten, indem Sie den Wert `spec.topology.nodePools.NAME.replicas` für den Ziel-Worker-Knotenpool bearbeiten.

Für einen v1alpha3-API-Cluster:

```
...
spec:
  topology:
    ...
    nodePools:
    - name: worker-1
      replicas: 3
...
```

```
...
spec:
  topology:
    ...
    nodePools:
    - name: worker-1
      replicas: 4
...
```

Für einen v1beta1-API-Cluster:

```
...
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  ...
spec:
  ...
  topology:
    ...
    class: tanzukubernetescluster
    controlPlane:
    ...
    workers:
      machineDeployments:
      - class: node-pool
        metadata: {}
        name: node-pool-1
        replicas: 3
...
```

```
...
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  ...
spec:
  ...
  topology:
    ...
    class: tanzukubernetescluster
    controlPlane:
    ...
    workers:
      machineDeployments:
      - class: node-pool
        metadata: {}
        name: node-pool-1
        replicas: 4
...
```

7 Um die Änderungen anzuwenden, speichern Sie die Datei im Texteditor. Um die Änderungen zu verwerfen, schließen Sie den Editor ohne zu speichern.

   Wenn Sie die Datei speichern, wendet kubectl die Änderungen auf den Cluster an. Der VM-Dienst im Supervisor stellt im Hintergrund den neuen Worker-Knoten bereit.

8 Vergewissern Sie sich, dass die neuen Knoten hinzugefügt wurden.

   Für einen v1alpha3-API-Cluster:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

Durch das horizontale Hochskalieren ergeben sich 4 Worker-Knoten für den Cluster.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster      3               4        v1.24.9---vmware.1-tkg.4
5d12h   True
```

Für einen v1beta1-API-Cluster:

```
kubectl get cluster tkg-cluster-1
```

Horizontales Herunterskalieren der Worker-Knoten

Sie können einen TKG-Cluster vertikal skalieren, indem Sie die Zahl der Worker-Knoten verringern.

1 Melden Sie sich bei Supervisor an.

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

2 Wechseln Sie den Kontext zum vSphere-Namespace, in dem der TKG-Cluster ausgeführt wird.

```
kubectl config use-context tkg-cluster-ns
```

3 Listet die Kubernetes-Cluster auf, die im vSphere-Namespace ausgeführt werden.

Verwenden Sie die folgende Syntax:

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

Beispielsweise für einen v1alph3-API-Cluster:

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

Beispielsweise für einen v1beta1-API-Cluster:

```
kubectl get cluster -n tkg-cluster-ns
```

4 Rufen Sie die Anzahl der im Zielcluster ausgeführten Knoten ab.

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

5 Rufen Sie die Anzahl der im Zielcluster ausgeführten Knoten ab.

Für einen v1alpha3-API-Cluster:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

Der folgende Cluster hat z. B. 3 Steuerungsebenenknoten und 4 Worker-Knoten.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster      3               4        v1.24.9---vmware.1-tkg.4
5d12h   True
```

Für einen v1beta1-API-Cluster:

```
kubectl get cluster tkg-cluster-1
```

6 Führen Sie zum Laden des Cluster-Manifests zur Bearbeitung den Befehl `kubectl edit` aus.

Für einen v1alpha3-API-Cluster:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

Für einen v1beta1-API-Cluster:

```
kubectl edit cluster/tkg-cluster-1
```

Das Cluster-Manifest wird in dem Texteditor geöffnet, der von den Umgebungsvariablen KUBE_EDITOR oder EDITOR definiert wird.

7 Verringern Sie die Anzahl der Worker-Knoten, indem Sie den Wert `spec.topology.nodePools.NAME.replicas` für den Ziel-Worker-Knotenpool bearbeiten.

Für einen v1alpha3-API-Cluster:

```
...
spec:
  topology:
    ...
    nodePools:
    - name: worker-1
      replicas: 4
...
```

```
...
spec:
  topology:
    ...
    nodePools:
    - name: worker-1
      replicas: 3
...
```

Für einen v1beta1-API-Cluster:

```
...
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  ...
spec:
  ...
  topology:
    ...
    class: tanzukubernetescluster
    controlPlane:
    ...
    workers:
      machineDeployments:
      - class: node-pool
        metadata: {}
        name: node-pool-1
        replicas: 4
...
```

```
...
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  ...
spec:
  ...
  topology:
    ...
    class: tanzukubernetescluster
    controlPlane:
    ...
    workers:
      machineDeployments:
      - class: node-pool
        metadata: {}
        name: node-pool-1
        replicas: 3
...
```

8 Um die Änderungen anzuwenden, speichern Sie die Datei im Texteditor. Um die Änderungen zu verwerfen, schließen Sie den Editor ohne zu speichern.

   Wenn Sie die Datei speichern, wendet kubectl die Änderungen auf den Cluster an. Der VM-Dienst im Supervisor stellt im Hintergrund den neuen Worker-Knoten bereit.

9 Vergewissern Sie sich, dass die Worker-Knoten entfernt wurden.

   Für einen v1alpha3-API-Cluster:

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

Durch das horizontale Herunterskalieren ergeben sich 3 Worker-Knoten für den Cluster.

```
NAMESPACE             NAME             CONTROL PLANE   WORKER   TKR NAME
AGE     READY
tkg-cluster-ns        tkg-cluster-1    3               3        v1.24.9---vmware.1-tkg.4
5d12h   True
```

Für einen v1beta1-API-Cluster:

```
kubectl get cluster tkg-cluster-1
```

Vertikale Skalierung eines Clusters

TKG auf Supervisor unterstützt vertikale Skalierung für Steuerungsebenen- und Worker-Knoten des Clusters. Sie führen die vertikale Skalierung eines TKG-Clusters durch, indem Sie die für Clusterknoten verwendete Verwenden von VM-Klassen mit TKG-Dienstclustern ändern. Die von Ihnen verwendete VM-Klasse muss an den vSphere-Namespace gebunden werden, in dem der TKG-Cluster bereitgestellt wird.

TKG auf Supervisor unterstützt vertikale Skalierung über den im System integrierten Mechanismus für parallele Aktualisierungen. Wenn Sie die `VirtualMachineClass`-Definition ändern, führt das System neue Knoten mit dieser Klasse ein und fährt die alten Knoten herunter. Weitere Informationen hierzu finden Sie unter Kapitel 9 Aktualisieren von TKG-Dienstclustern.

1 Melden Sie sich bei Supervisor an.

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

2 Wechseln Sie den Kontext zum vSphere-Namespace, in dem der TKG-Cluster ausgeführt wird.

```
kubectl config use-context tkg-cluster-ns
```

3 Listet die Kubernetes-Cluster auf, die im vSphere-Namespace ausgeführt werden.

Verwenden Sie die folgende Syntax:

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

Beispielsweise für einen v1alph3-API-Cluster:

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

Beispielsweise für einen v1beta1-API-Cluster:

```
kubectl get cluster -n tkg-cluster-ns
```

4 Beschreiben Sie den TKG-Zielcluster und überprüfen Sie die VM-Klasse.

Für einen v1alpha3-API-Cluster:

```
kubectl describe tanzukubernetescluster tkg-cluster-1
```

Der folgende Cluster verwendet z. B. die VM-Klasse „best-effort-medium“.

```
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: best-effort-medium
      ...
    nodePools:
    - name: worker-nodepool-a1
      replicas: 3
      vmClass: best-effort-medium
      ...
```

Für einen v1beta1-API-Cluster:

```
kubectl describe cluster tkg-cluster-1
```

Der folgende Cluster verwendet z. B. die VM-Klasse „best-effort-medium“.

```
...
Topology:
      ...
    Variables:
      ...
      Name:   vmClass
      Value:  best-effort-medium
    ...
```

Hinweis Standardmäßig ist die VM-Klasse für v1beta1-API-Cluster global als einzelne Variable festgelegt. Sie können diese Einstellung überschreiben und eine andere VM-Klasse für Steuerungsebenen- und Worker-Knoten verwenden. Weitere Informationen finden Sie unter vmClass in der API-Referenz.

5 Erstellen Sie eine Liste und eine Beschreibung der verfügbaren VM-Klassen.

```
kubectl get virtualmachineclass
```

```
kubectl describe virtualmachineclass
```

Hinweis Die VM-Klasse muss an den vSphere-Namespace gebunden werden. Weitere Informationen finden Sie unter Verwenden von VM-Klassen mit TKG-Dienstclustern.

6 Öffnen Sie zum Bearbeiten des Manifests des Zielclusters.

Für einen v1alpha3-API-Cluster:

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

Für einen v1beta1-API-Cluster:

```
kubectl edit cluster/tkg-cluster-1
```

Das Cluster-Manifest wird in dem Texteditor geöffnet, der von den Umgebungsvariablen KUBE_EDITOR oder EDITOR definiert wird.

7 Bearbeiten Sie das Manifest, indem Sie die VM-Klasse ändern.

Ändern Sie für einen v1alpha3-API-Cluster die VM-Klasse für die Steuerungsebene in `guaranteed-medium` und die VM-Klasse für Worker-Knoten in `guaranteed-large`.

```
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      ...
    nodePools:
    - name: worker-nodepool-a1
      replicas: 3
      vmClass: guaranteed-large
      ...
```

Ändern Sie für einen v1beta-API-Cluster die VM-Klasse in `guaranteed-large`.

```
...
Topology:
      ...
    Variables:
      ...
      Name:   vmClass
      Value:  guaranteed-large
    ...
```

8 Um die Änderungen anzuwenden, speichern Sie die Datei im Texteditor. Um die Änderungen zu verwerfen, schließen Sie den Editor ohne zu speichern.

Wenn Sie die Datei speichern, wendet kubectl die Änderungen auf den Cluster an. Im Hintergrund führt TKG auf Supervisor ein paralleles Update des TKG-Clusters durch.

9 Stellen Sie sicher, dass der TKG-Cluster mit der neuen VM-Klasse aktualisiert wurde.

Für einen v1alpha3-API-Cluster:

```
kubectl describe tanzukubernetescluster tkg-cluster-1
```

Für einen v1beta1-API-Cluster:

```
kubectl describe cluster tkg-cluster-1
```

Skalieren von Cluster-Knoten-Volumes

In der TKG-Clusterspezifikation für Knoten können Sie optional ein oder mehrere persistente Volumes für den Knoten deklarieren. Das Deklarieren eines Knoten-Volumes eignet sich für Komponenten mit hohen Änderungsraten, wie z. B. die Container-Laufzeit und kubelet auf Worker-Knoten.

Wenn Sie nach der Clustererstellung ein oder mehrere Knoten-Volumes hinzufügen oder ändern möchten, bedenken Sie Folgendes:

Volume-Knoten	Beschreibung
Änderungen des Worker-Knoten-Volumes sind zulässig	Nach der Bereitstellung eines TKG-Clusters können Sie Worker-Knoten-Volumes hinzufügen oder aktualisieren. Wenn Sie ein paralleles Update initiieren, wird der Cluster mit dem neuen oder geänderten Volume aktualisiert. **Warnung** Wenn Sie den Worker-Knoten mit einem neuen oder geänderten Volume skalieren, werden die Daten im aktuellen Volume während des parallelen Updates gelöscht. Weitere Informationen finden Sie in der folgenden Erläuterung. Ein für einen TKG-Clusterknoten deklariertes Volume wird als flüchtig behandelt. Ein TKG-Cluster verwendet eine Beanspruchung eines dauerhaften Volumes (Persistent Volume Claim, PVC) im vSphere Namespace, sodass die Volume-Kapazität dem Speicherkontingent des TKG-Clusters angerechnet wird. Wenn Sie die Kapazität eines TKC-Volumes erhöhen, stellt die Kubernetes-Cluster-API (CAPI) neue Worker mit einer neuen PVC bereit. TKG führt in diesem Fall keine Datenmigration durch. Kubernetes plant Arbeitslast-Pods entsprechend (neu).
Änderungen des Knoten-Volumes der Steuerungsebene sind zulässig, wenn Sie vSphere 8 U3 oder höher verwenden	Wenn Sie vSphere 8 U3 oder höher und ein kompatibles Tanzu Kubernetes Release verwenden, können Sie ein Knoten-Volume der Steuerungsebene nach der Bereitstellung eines TKG-Dienstclusters hinzufügen oder aktualisieren. Wenn Sie vSphere 8 U3 oder höher nicht verwenden, lässt die Kubernetes-Cluster-API (CAPI) keine Änderungen an `spec.toplogy.controlPlane.volumes` nach der Erstellung zu. Der Versuch, ein Steuerungsebenen-Volume nach der Clustererstellung hinzuzufügen oder zu ändern, wird zurückgewiesen, und Sie erhalten die Fehlermeldung „Aktualisierungen unveränderlicher Felder sind nicht zulässig“.

Bei Folgenden handelt es sich um einen Auszug aus einer Clusterspezifikation, die auf der v1alpha3-API mit deklarierten Knoten-Volumes basiert. Weitere Informationen finden Sie im vollständigen Beispiel eines TKG-Clusters, aus dem dieser Auszug nach Bedarf entnommen wird: v1alpha3-Beispiel: TKC mit Standardspeicher und Knoten-Volumes. Ein Beispiel für einen v1beta1-API-Cluster finden Sie unter v1beta1-Beispiel: Benutzerdefinierter Cluster basierend auf der standardmäßigen ClusterClass.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
...
spec:
   topology:
```

```
    controlPlane:
      replicas: 3
      storageClass: tkg-storage-policy
      vmClass: guaranteed-medium
      tkr:
        reference:
          name: v1.24.9---vmware.1-tkg.4
    nodePools:
    - name: worker-nodepool-a1
      replicas: 3
      storageClass: tkg-storage-policy
      vmClass: guaranteed-large
      tkr:
        reference:
          name: v1.24.9---vmware.1-tkg.4
      volumes:
      - name: containerd
        mountPath: /var/lib/containerd
        capacity:
          storage: 50Gi
      - name: kubelet
        mountPath: /var/lib/kubelet
        capacity:
          storage: 50Gi
    - name: worker-nodepool-a2
      ...
  settings:
    ...
```

Überwachen des TKG-Clusterstatus mithilfe des vSphere Client

Sie können den Status von TKG-Clustern mithilfe des vSphere Client überwachen.

Verfahren

1. Melden Sie sich über vSphere Client bei vCenter Server an.
2. Wählen Sie unter **Menü** die Ansicht **Hosts und Cluster** aus.
3. Erweitern Sie die Objekte **Datencenter > Cluster**, in denen der Supervisor erstellt wird.
4. Erweitern Sie den Ressourcenpool **Namespaces**.
5. Wählen Sie den vSphere-Namespace aus, in dem Sie den Tanzu Kubernetes-Cluster bereitgestellt haben.

   Jeder TKG-Cluster wird als Ordner innerhalb seines Namespace-Ressourcenpools aufgelistet. Jeder TKG-Cluster wird grafisch durch die drei Sechseck-Symbole neben seinem Namen dargestellt.
6. Wechseln Sie zur Ansicht **Menü > VMs und Vorlagen**.

   Innerhalb des Clusterordners sehen Sie die VMs, die die Clusterknoten beinhalten.

7 Wählen Sie den vSphere-Namespace und dann die Registerkarte **Berechnen** aus.

8 Wählen Sie unter **VMware-Ressourcen** die Option **Tanzu Kubernetes** aus.

Jeder in diesem vSphere-Namespace bereitgestellte TKG-Cluster ist aufgeführt.

Überwachen des TKG-Clusterstatus mithilfe von kubectl

Sie können den Status von bereitgestellten TKG-Clustern mit kubectl überwachen.

Verfahren

1 Authentifizieren Sie sich beim Supervisor.

2 Wechseln Sie zu dem vSphere-Namespace, in dem der Cluster ausgeführt wird.

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 Zeigen Sie eine Liste der Tanzu Kubernetes-Cluster an, die im Namespace ausgeführt werden.

```
kubectl get tanzukubernetesclusters
```

Dieser Befehl gibt den Status des Clusters zurück.

4 Zeigen Sie die Details des Clusters an.

```
kubectl describe tanzukubernetescluster <cluster-name>
```

Der Befehl gibt die Details des Clusters zurück. Im Abschnitt „Status“ der Befehlsausgabe werden ausführliche Informationen zum Cluster angezeigt.

```
...
Status:
  Addons:
    Cni:
      Name:    calico
      Status:  applied
    Csi:
      Name:    pvcsi
      Status:  applied
    Psp:
      Name:    defaultpsp
      Status:  applied
  Cloudprovider:
    Name:  vmware-guest-cluster
  Cluster API Status:
    API Endpoints:
      Host:  10.161.90.22
      Port:  6443
    Phase:   provisioned
  Node Status:
    test-tanzu-cluster-control-plane-0:                ready
    test-tanzu-cluster-workers-0-749458f97c-97ljv:     ready
```

```
  Phase:                                                running
  Vm Status:
    test-tanzu-cluster-control-plane-0:                 ready
    test-tanzu-cluster-workers-0-749458f97c-97ljv:      ready
Events:                                                 <none>
```

5 Führen Sie zusätzliche `kubectl`-Befehle aus, um weitere Details zum Cluster anzuzeigen. Weitere Informationen hierzu finden Sie unter Überprüfen von TKG-Clustervorgängen mithilfe von Kubectl.

Überprüfen der TKG-Clusterbereitschaft mithilfe von Kubectl

Wenn der TKG-Controller einen TKG-Cluster bereitstellt, werden mehrere Statusbedingungen gemeldet, die Sie verwenden können, um einen direkten Einblick in wichtige Aspekte der Computerintegrität zu erhalten.

Überprüfen der TKG-Clusterbereitschaft

Sie können mithilfe der Bedingungen für die TKG-Clusterbereitschaft ermitteln, welche Phase oder Komponente gegebenenfalls nicht bereit ist.

Nachdem Sie die Clusterbereitschaft überprüft haben, können Sie zur weiteren Diagnose die vSphereCluster- und Maschinenbedingungen verwenden, um detailliertere Fehler zu untersuchen.

So überprüfen Sie die Bereitschaft eines TKG-Clusters:

1 Melden Sie sich beim Supervisor an.

2 Ändern Sie den Kontext in den vSphere-Namespace, in dem der Zielcluster bereitgestellt wird. Beispiel:

```
kubectl config use-context tkg-cluster-ns
```

3 Führen Sie je nach Typ des TKG-Clusters den Befehl `kubectl get tkc -o yaml` oder `kubectl get cluster -o yaml` aus.

Der Befehl gibt die Bereitschaft der Clusterkomponenten zurück. Beschreibungen der verschiedenen Bereitschaftszustände finden Sie in den folgenden Abschnitten.

Bedingung und Gründe für ControlPlaneReady

In der Tabelle ist die `ControlPlaneReady`-Bedingung aufgeführt und beschrieben.

Tabelle 8-2. ControlPlaneReady-Bedingung

Bedingungstyp	Beschreibung
`ControlPlaneReady`	Meldet, ob die Steuerungsebenen-Knoten für den Cluster bereit und funktionsfähig sind.

In der Tabelle sind die Gründe aufgeführt und beschrieben, warum `ControlPlaneReady` „false“ sein kann.

Tabelle 8-3. Gründe für den ControlPlaneReady-Status „false“

Grund	Beschreibung
`WaitingForClusterInfrastructure`	Gibt an, dass der Cluster auf Voraussetzungen wartet, die für die Ausführung von Maschinen erforderlich sind, z. B. einen Load Balancer. Dieser Grund wird nur verwendet, wenn der InfrastructureCluster keine eigene „Bereit“-Bedingung meldet.
`WaitingForControlPlaneInitialized`	Gibt an, dass der erste Steuerungsebenen-Knoten initialisiert wird.
`WaitingForControlPlane`	Spiegelt den Zustand von KubeadmControlPlane wider. Dieser Grund wird verwendet, wenn KubeadmControlPlane keine eigene „Bereit“-Bedingung meldet.
Warten darauf, dass die Clusterinfrastruktur bereit ist	Meldung, die angibt, dass der Cluster auf die Voraussetzungen wartet, die für die Ausführung von Maschinen erforderlich sind, z. B. Netzwerk und Load Balancer.

Zustand und Gründe für „NodesHealthy“

In der Tabelle ist die `NodesHealthy`-Bedingung aufgeführt und beschrieben.

Tabelle 8-4. Zustand „NodesHealthy“

Bedingungstyp	Beschreibung
`NodesHealthy`	Meldet den Status der TanzuKubernetesCluster-Knoten.

In der Tabelle wird der Grund dafür beschrieben, warum die `NodesHealthy`-Bedingung nicht „true“ ist.

Tabelle 8-5. Grund für „NodesHealthy false“

Grund	Beschreibung
`WaitingForNodesHealthy`	Dokumentiert, dass nicht alle Knoten fehlerfrei sind.

Add-On-Bedingungen und -Gründe

In der Tabelle sind die Bedingungen im Zusammenhang mit Cluster-Add-On-Komponenten aufgeführt und beschrieben.

Tabelle 8-6. Add-On-Bedingungen

Bedingungstyp	Beschreibung
`AddonsReady`	Übersicht über die Bedingungen für TanzuKubernetesCluster-Add-Ons (CoreDNS, KubeProxy, CSP, CPI, CNI, AuthSvc).
`CNIProvisioned`	Dokumentiert den Status des Add-Ons TanzuKubernetesCluster CNI (Container Network Interface).
`CSIProvisioned`	Dokumentiert den Status des Add-Ons TanzuKubernetesCluster CSI (Container Storage Interface).
`CPIProvisioned`	Dokumentiert den Status des Add-Ons TanzuKubernetesCluster CPI (Cloud Provider Interface).
`KubeProxyProvisioned`	Dokumentiert den Status des Add-Ons TanzuKubernetesCluster KubeProxy.
`CoreDNSProvisioned`	Dokumentiert den Status des Add-Ons TanzuKubernetesCluster CoreDNS.
`AuthServiceProvisioned`	Dokumentiert den Status des Add-Ons TanzuKubernetesCluster AuthService.
`PSPProvisioned`	Dokumentiert den Status von PodSecurityPolicy.

In der Tabelle sind die Gründe aufgeführt, weshalb die Add-On-Bedingungen nicht zutreffen (nicht „true“ sind). Informationen zur Fehlerbehebung bei den Symptomen, die die Warnungen verursachen, finden Sie unter Kapitel 21 Fehlerbehebung bei TKG-Dienstclustern.

Tabelle 8-7. Gründe für den Add-On-Status „false“

Grund	Schweregrad	Beschreibung
`AddonsReconciliationFailed`	Nicht verfügbar	Zusammengefasster Grund für alle Add-On-Abgleichfehler.
`CNIProvisioningFailed`	Warnung	Dokumentiert, dass das CNI-Add-On nicht erstellt oder aktualisiert werden konnte.
`CSIProvisioningFailed`	Warnung	Dokumentiert, dass das CSI-Add-On nicht erstellt oder aktualisiert werden konnte.
`CPIProvisioningFailed`	Warnung	Dokumentiert, dass das CPI-Add-On nicht erstellt oder aktualisiert werden konnte.
`KubeProxyProvisioningFailed`	Warnung	Dokumentiert, dass das KubeProxy-Add-On nicht erstellt oder aktualisiert werden konnte.
`CoreDNSProvisioningFailed`	Warnung	Dokumentiert, dass das CoreDNS-Add-On nicht erstellt oder aktualisiert werden konnte.

Tabelle 8-7. Gründe für den Add-On-Status „false“ (Fortsetzung)

Grund	Schweregrad	Beschreibung
`AuthServiceProvisioningFailed`	Warnung	Dokumentiert, dass das AuthService-Add-On nicht erstellt oder aktualisiert werden konnte.
`AuthServiceUnManaged`		Der Dokumentauthentifizierungsdienst wird nicht vom Controller verwaltet.
`PSPProvisioningFailed`	Warnung	Dokumentiert, dass die PodSecurityPolicy-Add-Ons nicht erstellt oder aktualisiert werden konnten.

Andere Bedingungen und Gründe

In der Tabelle sind die Bedingungen für die StorageClass- und RoleBinding-Synchronisierung, den ProviderServiceAccount-Ressourcenabgleich, ServiceDiscovery und die TKG 2.0-Clusterkompatibilität aufgeführt.

Tabelle 8-8. Andere Bedingungen

Zustand	Beschreibung
`StorageClassSynced`	Dokumentiert den Status der StorageClass-Synchronisierung vom Supervisor-Cluster zum Arbeitslastcluster.
`RoleBindingSynced`	Dokumentiert den Status der RoleBinding-Synchronisierung vom Supervisor-Cluster zum Arbeitslastcluster.
`ProviderServiceAccountsReady`	Dokumentiert den Status von Anbieterdienstkonten und zugehörige Rollen, RoleBindings und geheime Schlüssel werden erstellt.
`ServiceDiscoveryReady`	Dokumentiert den Status der Dienst-Entdeckungen.
`TanzuKubernetesReleaseCompatible`	Gibt an, ob der TanzuKubernetesCluster mit dem TanzuKubernetesRelease kompatibel ist.

In der Tabelle sind die Gründe aufgeführt, warum andere Bedingungen nicht zutreffen (nicht „true“ sind).

Tabelle 8-9. Andere Gründe

Grund	Beschreibung
`StorageClassSyncFailed`	Meldet, dass die StorageClass-Synchronisierung fehlgeschlagen ist.
`RoleBindingSyncFailed`	Meldet, dass die RoleBinding-Synchronisierung fehlgeschlagen ist.

Tabelle 8-9. Andere Gründe (Fortsetzung)

Grund	Beschreibung
`ProviderServiceAccountsReconciliationFailed`	Meldet, dass der Abgleich der anbieterdienstbezogenen Ressourcen fehlgeschlagen ist.
`SupervisorHeadlessServiceSetupFailed`	Dokumentiert, dass die Headless-Diensteinrichtung für Supervisor Cluster API-Server fehlgeschlagen ist.

Überprüfen der Integrität der TKG-Clustermaschine mithilfe von Kubectl

Wenn der TKG-Controller einen Arbeitslast-Cluster auf Supervisor bereitstellt, werden mehrere Statusbedingungen gemeldet, die Sie verwenden können, um einen direkten Einblick in wichtige Aspekte der Integrität der Maschine zu erhalten.

Informationen zu den Bedingungen der Computerintegrität

Ein TKG-Cluster umfasst mehrere bewegliche Teile, die alle von unabhängigen, aber zugehörigen Controllern gesteuert werden und zusammenarbeiten, um einen Satz von Kubernetes-Knoten zu erstellen und zu pflegen. Das `TanzuKubernetesCluster`- und das `Cluster`-Objekt stellen Statusbedingungen bereit, mit denen Sie detaillierte Informationen zur Computerintegrität erhalten.

Überprüfen der Computerintegrität

So überprüfen Sie die Integrität einer TKG-Clustermaschine:

1 Stellen Sie eine Verbindung mit Supervisor her und melden Sie sich an.

2 Ändern Sie den Kontext in den vSphere-Namespace, in dem der TKG-Zielcluster bereitgestellt wird.

```
kubectl config use-context CLUSTER-NAME
```

3 Führen Sie den Befehl `kubectl describe machine` aus.

Der Befehl gibt den Status der VM-Knoten zurück, aus denen sich der Cluster zusammensetzt. Wenn eine Maschinenbedingung wie `InfrastructureReady` `True` und `Ready` ist, ist dieser Aspekt der Maschine fehlerfrei. Wenn jedoch eine Maschinenbedingung `False` ist, ist die Maschine nicht bereit. Beschreibungen der einzelnen Maschinenbedingungstypen finden Sie in der Liste der Bedingungen für die Maschinenintegrität.

4 Wenn die Maschine nicht bereit ist, führen Sie den folgenden Befehl aus und ermitteln Sie, was mit der Infrastruktur nicht stimmt:

```
kubectl describe vspheremachine
```

Liste der Betriebszustände der Maschine

In der Tabelle sind die verfügbaren Maschinenintegritätsbedingungen für einen TKG-Cluster aufgelistet und definiert.

Zustand	Beschreibung
`ResourcePolicyReady`	Meldet die erfolgreiche Erstellung einer Ressourcenrichtlinie.
`ResourcePolicyCreationFailed`	Wird gemeldet, wenn während der Erstellung von Ressourcenrichtlinien Fehler auftreten.
`ClusterNetworkReady`	Meldet die erfolgreiche Bereitstellung eines Clusternetzwerks.
`ClusterNetworkProvisionStarted`	Wird gemeldet, wenn auf die Bereitschaft des Clusternetzwerks gewartet wird.
`ClusterNetworkProvisionFailed`	Wird gemeldet, wenn während der Netzwerkbereitstellung Fehler auftreten.
`LoadBalancerReady`	Meldet die erfolgreiche Abstimmung eines statischen Endpunkts auf Steuerungsebene.
`LoadBalancerCreationFailed`	Wird gemeldet, wenn die Erstellung von Lastausgleichsdienst-Ressourcen fehlschlägt.
`WaitingForLoadBalancerIP`	Wird gemeldet, wenn auf die IP-Adresse des Lastausgleichsdiensts gewartet wird.
`VMProvisioned`	Meldet, dass eine virtuelle Maschine erstellt, eingeschaltet und eine IP-Adresse zugewiesen ist.
`WaitingForBootstrapData`	Wird gemeldet, wenn ein vSphere-Maschine auf die Bereitschaft des Bootstrap-Skripts wartet, bevor der Bereitstellungsprozess gestartet wird.
`VMCreationFailed`	Meldet, dass die Erstellung von VM CRD oder der entsprechenden Bootstrap-ConfigMap fehlgeschlagen ist.
`VMProvisionStarted`	Wird gemeldet, wenn eine virtuelle Maschine derzeit erstellt wird.
`PoweringOn`	Wird gemeldet, wenn eine virtuelle Maschine derzeit den Einschaltprozess ausführt.
`WaitingForNetworkAddress`	Wird gemeldet, wenn gewartet wird, bis die Netzwerkeinstellungen der Maschine aktiv werden.
`WaitingForBIOSUUID`	Wird gemeldet, wenn gewartet wird, bis die Maschine über eine BIOS UUID verfügt.

Zustandsfelder

Jede Bedingung kann mehrere Felder enthalten.

`Type`	Beschreibt den Typ des Zustands. Beispielsweise `ResourcePolicyReady`. Beim Zustand `Ready` handelt es sich um eine Zusammenfassung aller anderen Zustände.
`Status`	Beschreibt den Status des Typs. Zustände können `True`, `False` oder `Unknown` sein.
`Severity`	Einstufung des `Reason`. `Info` bedeutet, dass die Abstimmung stattfindet. `Warning` bedeutet, dass etwas falsch ist und erneut versucht wird. `Error` bedeutet, dass ein Fehler aufgetreten ist und manuelle Eingriffe für die Lösung erforderlich sind.
`Reason`	Liefert einen Grund, weshalb der Status `False` ist. Es kann sich um einen Grund „Warten auf Bereitschaft“ oder um eine Fehlerursache handeln. Wird in der Regel ausgelöst, wenn der Status `False` lautet.
`Message`	Von Menschen lesbare Informationen, die den `Reason` erläutert.

Überprüfen der Integrität des TKG-Clusters mithilfe von Kubectl

Wenn der TKG-Controller einen Arbeitslast-Cluster bereitstellt, werden mehrere Statusbedingungen gemeldet, die Sie verwenden können, um einen direkten Einblick in wichtige Aspekte der Clusterintegrität zu erhalten.

Informationen zu den Bedingungen der Cluster-Integrität

Ein bereitgestellter TKG-Cluster umfasst mehrere bewegliche Teile, die alle von unabhängigen, aber zugehörigen Controllern gesteuert werden und zusammenarbeiten, um einen Satz von Kubernetes-Knoten zu erstellen und zu pflegen. Die `TanzuKubernetesCluster`- und `Cluster`-Objekte stellen Statusbedingungen bereit, mit denen Sie detaillierte Informationen zur Integrität des Clusters und des Computers erhalten.

Integrität des Clusters überprüfen

So überprüfen Sie die Integrität eines TKG-Clusters:

1 Führen Sie den Befehl `kubectl describe cluster` aus.

Wenn der Status „Bereit“ ist, bedeutet dies, dass sowohl die Clusterinfrastruktur als auch die Cluster-Steuerungskomponente bereit sind. Beispiel:

```
Status:
  Conditions:
    Last Transition Time:     2020-11-24T21:37:32Z
    Status:                   True
    Type:                     Ready
    Last Transition Time:     2020-11-24T21:37:32Z
    Status:                   True
```

```
    Type:                    ControlPlaneReady
    Last Transition Time:    2020-11-24T21:31:34Z
    Status:                  True
    Type:                    InfrastructureReady
```

Wenn jedoch eine Clusterbedingung „false“ lautet, ist der Cluster nicht bereit, und in einem Nachrichtenfeld wird beschrieben, wo der Fehler liegt. Beispielsweise lautet hier der Status „false“ und der Grund ist, dass die Infrastruktur nicht bereit ist:

```
Status:
  Conditions:
    Last Transition Time:    2020-11-24T21:37:32Z
    Status:                  False
    Type:                    Ready
    Last Transition Time:    2020-11-24T21:37:32Z
    Status:                  True
    Type:                    ControlPlaneReady
    Last Transition Time:    2020-11-24T21:31:34Z
    Status:                  False
    Type:                    InfrastructureReady
```

2 Wenn der Cluster nicht bereit ist, führen Sie den folgenden Befehl aus, um festzustellen, was mit der Clusterinfrastruktur nicht stimmt:

```
kubectl describe vspherecluster
```

Liste der Bedingungen der Cluster-Integrität

In der Tabelle sind die verfügbaren Integritätsbedingungen für einen TKG-Cluster aufgelistet und definiert.

Bedingung	Beschreibung
`Ready`	Fasst den Betriebszustand eines Cluster-API-Objekts zusammen.
`Deleting`	Der Status ist nicht „True“, da das zugrunde liegende Objekt derzeit gelöscht wird.
`DeletionFailed`	Der Status ist nicht „True“, da beim zugrunde liegenden Objekt Probleme während des Löschens aufgetreten sind. Es handelt sich hierbei um eine Warnung, da der Abstimmungsalgorithmus den Löschvorgang erneut versucht.
`Deleted`	Der Status ist nicht „True“, da das zugrunde liegende Objekt gelöscht wurde.
`InfrastructureReady`	Meldet eine Übersicht über den aktuellen Status des für diesen Cluster definierten Infrastrukturobjekts.

Bedingung	Beschreibung
`WaitingForInfrastructure`	Wird gemeldet, wenn ein Cluster wartet, bis die zugrunde liegende Infrastruktur verfügbar ist. HINWEIS: Dieser Zustand wird als Fallback verwendet, wenn die Infrastruktur keinen Status „Bereit“ meldet.
`ControlPlaneReady`	Wird gemeldet, wenn die Clustersteuerungskomponente bereit ist.
`WaitingForControlPlane`	Wird gemeldet, wenn ein Cluster wartet, bis die Steuerungsebene verfügbar ist. HINWEIS: Dieser Zustand wird als Fallback verwendet, wenn die Steuerungsebene keinen Status „Bereit“ meldet.

Zustandsfelder

Jede Bedingung kann mehrere Felder enthalten.

`Type`	Beschreibt den Typ des Zustands. Beispielsweise `ControlPlaneReady`. Beim Zustand `Ready` handelt es sich um eine Zusammenfassung aller anderen Zustände.
`Status`	Beschreibt den Status des Typs. Zustände können `True`, `False` oder `Unknown` sein.
`Severity`	Einstufung des `Reason`. `Info` bedeutet, dass die Abstimmung stattfindet. `Warning` bedeutet, dass etwas falsch ist und erneut versucht wird. `Error` bedeutet, dass ein Fehler aufgetreten ist und manuelle Eingriffe für die Lösung erforderlich sind.
`Reason`	Liefert einen Grund, weshalb der Status `False` ist. Es kann sich um einen Grund „Warten auf Bereitschaft“ oder um eine Fehlerursache handeln. Wird in der Regel ausgelöst, wenn der Status `False` lautet.
`Message`	Von Menschen lesbare Informationen, die den `Reason` erläutert.

Überprüfen der Integrität des TKG-Cluster-Volumes mithilfe von Kubectl

Sie können den Integritätsstatus eines dauerhaften Volumes in einem gebundenen Zustand in einem TKG-Cluster überprüfen.

Für jedes persistente Volume in einem gebundenen Zustand wird der Integritätsstatus im Feld `Annotations: volumehealth.storage.kubernetes.io/messages:` der Beanspruchung eines dauerhaften Volumes angezeigt, das an das dauerhafte Volume gebunden ist. Es gibt zwei mögliche Werte für den Integritätsstatus.

Integritätsstatus	Beschreibung
Verfügbar	Auf das dauerhafte Volume kann zugegriffen werden, und es steht zur Verwendung zur Verfügung.
Kein Zugriff	Auf das dauerhafte Volume kann nicht zugegriffen werden, und es kann nicht verwendet werden. Auf das dauerhafte Volume kann nicht zugegriffen werden, wenn der Datenspeicher, auf dem das Volume gespeichert wird, nicht von den Hosts erreicht werden kann, die eine Verbindung zum Datenspeicher herstellen.

Verfahren

1 Melden Sie sich über Kubectl bei Ihrem TKG-Cluster an.

2 Erstellen Sie einen Anspruch für dauerhafte Volumes.

a Erstellen Sie eine YAML-Datei, die die Konfiguration der Beanspruchung eines dauerhaften Volumes enthält.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gold
  resources:
    requests:
        storage: 2Gi
```

b Wenden Sie die Beanspruchung eines dauerhaften Volumes auf den Kubernetes-Cluster an.

```
kubectl apply -f pvc_name.yaml
```

Dieser Befehl erstellt ein dauerhaftes Kubernetes-Volume sowie ein vSphere-Volume mit einer zugrunde liegenden virtuellen Festplatte, die die Speicheranforderungen des Anspruchs erfüllt.

c Überprüfen Sie, ob die Beanspruchung eines dauerhaften Volumes an ein Volume gebunden ist.

```
kubectl get pvc my-pvc
```

Die Ausgabe zeigt, dass sich die Beanspruchung eines dauerhaften Volumes und das Volume im gebundenen Zustand befinden.

```
NAME     STATUS    VOLUME   CAPACITY   ACCESSMODES   STORAGECLASS   AGE
my-pvc   Bound     my-pvc   2Gi        RWO           gold           30s
```

3 Überprüfen Sie den Integritätsstatus des Volumes.

Führen Sie den folgenden Befehl aus, um die Volume-Integritätsanmerkung der Beanspruchung eines persistenten Volumes zu überprüfen, die an das dauerhafte Volume gebunden ist.

```
kubectl describe pvc my-pvc
```

In der folgenden Beispielausgabe weist das Feld `volumehealth.storage.kubernetes.io/messages` den Integritätsstatus als „Verfügbar“ aus.

```
Name:          my-pvc
Namespace:     test-ns
StorageClass:  gold
Status:        Bound
Volume:        my-pvc
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner: csi.vsphere.vmware.com
               volumehealth.storage.kubernetes.io/messages: accessible
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      2Gi
Access Modes:  RWO
VolumeMode:    Filesystem
```

Überwachen der Volume-Integrität in einem Tanzu Kubernetes Grid-Cluster

Sie können den Integritätsstatus eines dauerhaften Volumes in einem gebundenen Zustand überprüfen.

Für jedes persistente Volume in einem gebundenen Zustand wird der Integritätsstatus im Feld `Annotations: volumehealth.storage.kubernetes.io/messages:` der Beanspruchung eines dauerhaften Volumes angezeigt, das an das dauerhafte Volume gebunden ist. Es gibt zwei mögliche Werte für den Integritätsstatus.

Integritätsstatus	Beschreibung
Verfügbar	Auf das dauerhafte Volume kann zugegriffen werden, und es steht zur Verwendung zur Verfügung.
Kein Zugriff	Auf das dauerhafte Volume kann nicht zugegriffen werden, und es kann nicht verwendet werden. Auf das dauerhafte Volume kann nicht zugegriffen werden, wenn der Datenspeicher, auf dem das Volume gespeichert wird, nicht von den Hosts erreicht werden kann, die eine Verbindung zum Datenspeicher herstellen.

Verfahren

1 Greifen Sie in der vSphere Kubernetes-Umgebung auf Ihren Namespace zu.

2 Erstellen Sie einen Anspruch für dauerhafte Volumes.

a Erstellen Sie eine YAML-Datei, die die Konfiguration der Beanspruchung eines dauerhaften Volumes enthält.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gold
  resources:
    requests:
        storage: 2Gi
```

b Wenden Sie die Beanspruchung eines dauerhaften Volumes auf den Kubernetes-Cluster an.

```
kubectl apply -f pvc_name.yaml
```

Dieser Befehl erstellt ein dauerhaftes Kubernetes-Volume sowie ein vSphere-Volume mit einer zugrunde liegenden virtuellen Festplatte, die die Speicheranforderungen des Anspruchs erfüllt.

c Überprüfen Sie, ob die Beanspruchung eines dauerhaften Volumes an ein Volume gebunden ist.

```
kubectl get pvc my-pvc
```

Die Ausgabe zeigt, dass sich die Beanspruchung eines dauerhaften Volumes und das Volume im gebundenen Zustand befinden.

```
NAME     STATUS    VOLUME   CAPACITY   ACCESSMODES   STORAGECLASS   AGE
my-pvc   Bound     my-pvc   2Gi        RWO           gold           30s
```

3 Überprüfen Sie den Integritätsstatus des Volumes.

Führen Sie den folgenden Befehl aus, um die Volume-Integritätsanmerkung der Beanspruchung eines persistenten Volumes zu überprüfen, die an das dauerhafte Volume gebunden ist.

```
kubectl describe pvc my-pvc
```

In der folgenden Beispielausgabe weist das Feld `volumehealth.storage.kubernetes.io/messages` den Integritätsstatus als „Verfügbar“ aus.

```
Name:          my-pvc
Namespace:     test-ns
StorageClass:  gold
```

```
Status:        Bound
Volume:        my-pvc
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner: csi.vsphere.vmware.com
               volumehealth.storage.kubernetes.io/messages: accessible
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      2Gi
Access Modes:  RWO
VolumeMode:    Filesystem
```

Überwachen von dauerhaften Volumes mithilfe des vSphere Client

Wenn DevOps-Ingenieure eine statusbehaftete Anwendung mit einem Anspruch für dauerhafte Volumes bereitstellen, erstellt vSphere IaaS control plane ein dauerhaftes Volume-Objekt und eine passende dauerhafte virtuelle Festplatte. Als vSphere-Administrator können Sie Details des dauerhaften Volumes im vSphere Client überprüfen. Sie können auch die Speicherübereinstimmung und den Systemzustand des Volumes überwachen.

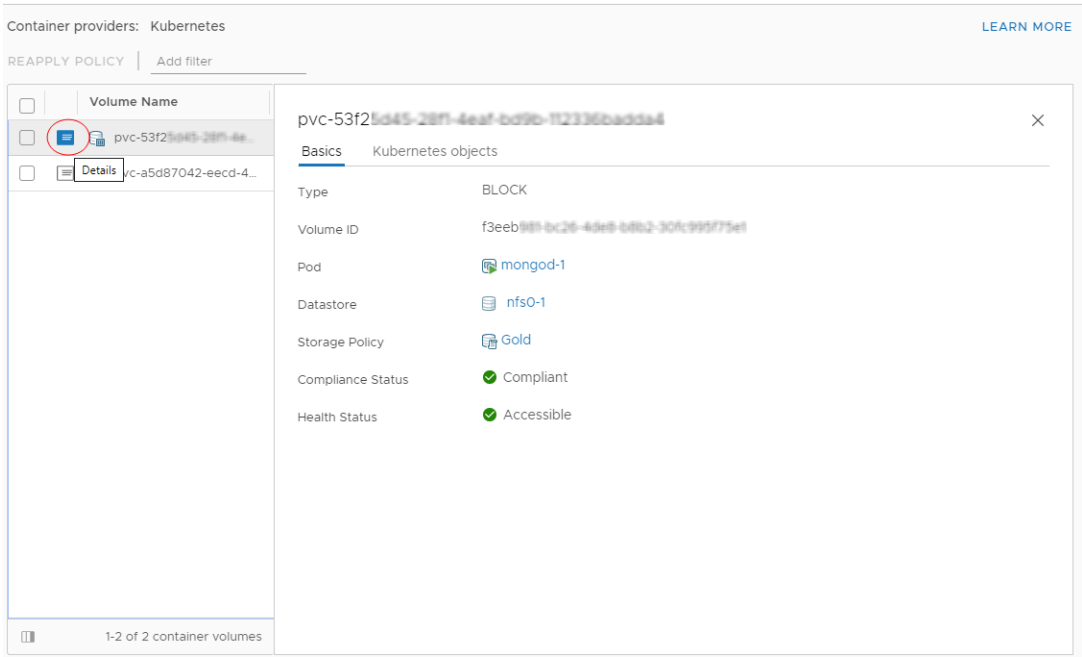
Verfahren

1. Navigieren Sie im vSphere Client zu dem Namespace, der über dauerhafte Volumes verfügt.
   a. Wählen Sie im vSphere Client-Startmenü die Option **Arbeitslastverwaltung** aus.
   b. Klicken Sie auf die Registerkarte **Namespaces** und wählen Sie einen Namespace aus der Liste aus.
2. Klicken Sie auf die Registerkarte **Speicher** und dann auf **Anforderungen von dauerhaften Datenträgern**.

   Der vSphere Client listet alle Objekte mit Ansprüchen für dauerhafte Volumes sowie entsprechende im Namespace verfügbare Volumes auf.
3. Klicken Sie zum Anzeigen der Details eines Anspruchs für ein ausgewähltes dauerhaftes Volume in der Spalte **Name des dauerhaften Volumes** auf den Namen des Volumes.

4 Überprüfen Sie auf der Seite **Container-Volumes** den Integritätsstatus und die Speicherrichtlinienübereinstimmung des Volumes.

a Klicken Sie auf das Symbol **Details** und wechseln Sie zwischen den Registerkarten **Grundlagen** und **Kubernetes-Objekte**, um zusätzliche Informationen für das dauerhafte Kubernetes-Volume anzuzeigen.



b Überprüfen Sie den Integritätsstatus des Volumes.

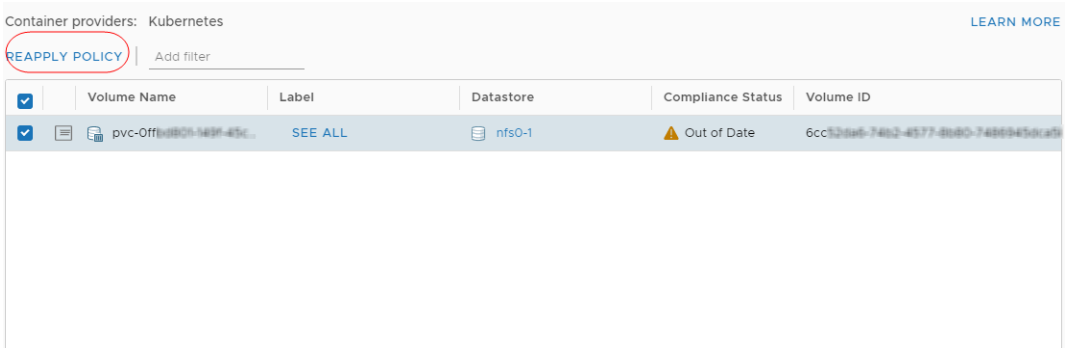
Integritätsstatus	Beschreibung
Verfügbar	Auf das dauerhafte Volume kann zugegriffen werden, und es steht zur Verwendung zur Verfügung.
Kein Zugriff	Auf das dauerhafte Volume kann nicht zugegriffen werden, und es kann nicht verwendet werden. Auf das dauerhafte Volume kann nicht zugegriffen werden, wenn der Datenspeicher, auf dem das Volume gespeichert wird, nicht von den Hosts erreicht werden kann, die eine Verbindung zum Datenspeicher herstellen.

c Überprüfen Sie den Status der Speicherübereinstimmung.

Einer der folgenden Werte wird in der Spalte **Übereinstimmungsstatus** angezeigt.

Übereinstimmungsstatus	Beschreibung
Übereinstimmung	Der Datenspeicher, in dem sich die zugrunde liegende virtuelle Festplatte des Volumes befindet, enthält die von der Richtlinie benötigten Speicherfunktionen.
Veraltet	Dieser Status gibt an, dass die Richtlinie bearbeitet wurde, die neuen Anforderungen aber nicht an den Datenspeicher weitergegeben wurden. Zum Übermitteln der Änderungen wenden Sie die Richtlinie erneut auf das veraltete Volume an.
Nicht übereinstimmend	Der Datenspeicher unterstützt festgelegte Speicheranforderungen, kann jedoch zurzeit nicht die Speicherrichtlinie erfüllen. Der Status kann z. B. in „Keine Übereinstimmung“ wechseln, wenn physische Ressourcen für den Datenspeicher nicht verfügbar sind. Sie können die Übereinstimmung des Datenspeichers wiederherstellen, indem Sie Änderungen an der physischen Konfiguration Ihres Hostclusters vornehmen, indem Sie Hosts oder Festplatten beispielsweise zum Cluster hinzufügen. Wenn weitere Ressourcen die Speicherrichtlinie erfüllen, ändert sich der Status in „Übereinstimmung“.
Nicht anwendbar	Die Speicherrichtlinie verweist auf Datenspeicherkapazitäten, die vom Datenspeicher nicht unterstützt werden.

d Wenn der Übereinstimmungsstatus auf „Veraltet“ gesetzt ist, wählen Sie das Volume aus und klicken Sie auf **Richtlinie erneut anwenden**.



Der Status wird in „Übereinstimmung“ geändert.

Abrufen von geheimen Schlüsseln für TKG-Cluster mithilfe von Kubectl

TKG-Cluster verwenden geheime Schlüssel zum Speichern von Token, Schlüsseln und Kennwörtern für den Betrieb von.

Liste der geheimen Schlüssel für TKG-Cluster

Ein geheimer Kubernetes-Schlüssel ist ein Objekt, in dem eine kleine Menge sensibler Daten gespeichert wird, beispielsweise ein Kennwort, ein Token oder ein SSH-Schlüssel. TKG-Clusteradministratoren können beim Betrieb von Clustern mehrere geheime Schlüssel verwenden. In der Tabelle werden geheime Schlüssel aufgelistet und beschrieben, die Clusteradministratoren möglicherweise verwenden.

Hinweis Die Liste ist nicht vollständig. Sie enthält nur die geheimen Schlüssel, die möglicherweise manuell rotiert oder für den Zugriff auf Clusterknoten zu Fehlerbehebungszwecken verwendet werden.

Geheim	Beschreibung
TANZU-KUBERNETES-CLUSTER-NAME-ccm-token-RANDOM	Ein Dienstkonto-Token, das vom Cloud-Controller-Manager des Anbieters paravirtueller Clouds verwendet wird, um eine Verbindung mit dem vSphere-Namespace herzustellen. Löschen Sie den geheimen Schlüssel, um die Rotation dieser Anmeldedaten auszulösen.
TANZU-KUBERNETES-CLUSTER-NAME-pvcsi-token-RANDOM	Ein Dienstkonto-Token, das vom paravirtuellen CSI-Plug-In verwendet wird, um eine Verbindung mit dem vSphere-Namespace herzustellen. Löschen Sie den geheimen Schlüssel, um die Rotation dieser Anmeldedaten auszulösen.
TANZU-KUBERNETES-CLUSTER-NAME-kubeconfig	Eine kubeconfig-Datei, mit der es möglich ist, als Benutzer `kubernetes-admin` eine Verbindung mit der Steuerungsebene des Clusters herzustellen. Dieser geheime Schlüssel kann verwendet werden, um auf einen Cluster zuzugreifen und Fehler in ihm zu beheben, wenn die vCenter Single Sign-On-Authentifizierung nicht verfügbar ist.
TANZU-KUBERNETES-CLUSTER-NAME-ssh	Ein privater SSH-Schlüssel, mit dem es möglich ist, als `vmware-system-user` eine Verbindung mit jedem Clusterknoten herzustellen. Dieser geheime Schlüssel kann zur Anmeldung bei jedem Clusterknoten per SSH und zur Fehlerbehebung darauf verwendet werden.
TANZU-KUBERNETES-CLUSTER-NAME-ssh-password	Ein Kennwort, mit dem es möglich ist, als `vmware-system-user` eine Verbindung mit jedem Clusterknoten herzustellen.
TANZU-KUBERNETES-CLUSTER-NAME-ca	Das Root-CA-Zertifikat für die Tanzu Kubernetes-Cluster-Steuerungsebene, das von `kubectl` zum Herstellen einer sicheren Verbindung mit dem Kubernetes-API-Server verwendet wird.

Überprüfen des TKG-Clusternetzwerks mithilfe von Kubectl

Das System stellt TKG-Cluster mit Standardnetzwerken für Knoten, Pods und Dienste bereit. Sie können das Cluster-Netzwerk mithilfe von benutzerdefinierten kubectl-Befehlen überprüfen.

Benutzerdefinierte Befehle zur Überprüfung des TKG-Clusternetzwerks

Verwenden Sie die folgenden Befehle zum Überprüfen des Cluster-Netzwerks.

Diese Befehle sollten in dem vSphere-Namespace ausgeführt werden, in dem der TKG-Cluster bereitgestellt wird. Beispiel:

```
kubectl config use-context tkg2-cluster-ns
```

Tabelle 8-10. Benutzerdefinierte kubectl-Befehle zum Überprüfen des Cluster-Netzwerks

Befehl	Beschreibung

Befehl
`kubectl get tkgserviceconfigurations`
Beispielergebnis
`NAME DEFAULT CNI`
`tkg-service-configuration antrea`
	Gibt das Standard-CNI `antrea` zurück, sofern nicht geändert.
	Das Standard-CNI wird für die Clustererstellung verwendet, sofern dies nicht explizit in der Cluster-YAML überschrieben wurde.

Befehl
`kubectl get virtualnetwork`
Beispielergebnis
`NAME SNAT`
`READY AGE`
`tkgs-cluster-12-vnet 10.191.152.133`
`True 4h3m`
	Gibt das virtuelle Netzwerk für Clusterknoten zurück.
	Verwenden Sie diesen Befehl, um zu überprüfen, ob die IP-Adresse für die Quell-Netzwerkadressübersetzung (Source Network Address Translation, SNAT) zugewiesen ist.

Befehl
`kubectl get virtualmachines -o wide`
Beispielergebnis
`NAME`
`POWERSTATE CLASS`
`IMAGE`
`PRIMARY-IP AGE`
`tkg2-cluster-12-control-plane-...`
`poweredOn guaranteed-medium`
`ob-...-v1.23.8---vmware.1-tkg.1.b3d708a`
`10.244.0.66 4h6m`
`tkg2-cluster-12-worker-...`
`poweredOn guaranteed-medium`
`ob-...-v1.22.9---vmware.1-tkg.1.b3d708a`
`10.244.0.68 4h3m`
`tkg2-cluster-12-worker-...`
`poweredOn guaranteed-medium`
`ob-...-v1.21.6---vmware.1-tkg.1.b3d708a`
`10.244.0.67 4h3m`
	Gibt die virtuelle Netzwerkschnittstelle für Clusterknoten zurück.
	Verwenden Sie diesen Befehl, um zu überprüfen, ob der virtuellen Maschine für jeden Clusterknoten eine IP-Adresse zugewiesen wurde.

Tabelle 8-10. Benutzerdefinierte kubectl-Befehle zum Überprüfen des Cluster-Netzwerks (Fortsetzung)

Befehl	Beschreibung

Befehl
`kubectl get virtualmachineservices`
Beispielergebnis
`NAME TYPE AGE tkg2-cluster-12-control-plane-service LoadBalancer 3h53m`
	Gibt den VM-Dienst für jeden Clusterknoten zurück.
	Verwenden Sie diesen Befehl, um zu überprüfen, ob der Status aktualisiert wurde und die virtuelle IP-Adresse (VIP) des Load Balancers umfasst.

Befehl
`kubectl get services -n NAMESPACE`
Überprüfung mithilfe von cURL
`curl -k https://EXTERNAL-IP:PORT/healthz`
	Gibt den für den Cluster-API-Zugriff erstellten Load Balancer des Kubernetes-Dienstes zurück. Verwenden Sie diesen Befehl, um zu überprüfen, ob eine externe IP zugewiesen ist.
	Überprüfen Sie mit `curl`, ob über die externe IP-Adresse und den Port des Lastausgleichsdiensts Zugriff auf die API besteht.

Befehl
`kubectl get endpoints`
Beispielergebnis
`NAME ENDPOINTS AGE tkg2-cluster-12-control-plane-service 10.244.0.66:6443 3h44m`
	Gibt die Steuerungsebenenknoten (Endpoints) für den Cluster zurück. Verwenden Sie diesen Befehl, um zu überprüfen, ob jeder Endpoint erstellt und in den Endpoint-Pool eingeschlossen wurde.

Überprüfen von TKG-Clustervorgängen mithilfe von Kubectl

Sie können TKG-Cluster mithilfe von benutzerdefinierten kubectl-Befehlen verwalten. Diese Befehle werden durch benutzerdefinierte Ressourcen zur Verfügung gestellt, die vom TKG-Controller verwaltet werden.

Benutzerdefinierte Befehle zum Verwalten von TKG-Clustern

In der Tabelle werden die kubectl-Befehle zur Verwaltung von TKG-Clustern aufgeführt und beschrieben.

Führen Sie jeden Befehl im Kontext für den vSphere-Namespace aus, in dem der TKG-Cluster bereitgestellt wird. Durch die Ausführung dieser Befehle im Clusterkontext werden keine Informationen zurückgegeben.

Tabelle 8-11. Benutzerdefinierte Befehle zum Verwalten von TKG-Clustern

Befehl	Beschreibung
`kubectl get tanzukubernetescluster`	Listet die TKCs im aktuellen Namespace auf.
`kubectl get tkc`	Kurzformversion des vorhergehenden Befehls.
`kubectl get cluster`	Gibt Cluster im Namespace zurück.
`kubectl describe tanzukubernetescluster CLUSTER-NAME`	Beschreibt den angegebenen Cluster und zeigt den angegebenen Zustand, den Status und die Ereignisse an. Nach Abschluss der Bereitstellung zeigt dieser Befehl die virtuelle IP an, die für den Load Balancer erstellt wurde, der den Kubernetes-API-Endpoints vorgelagert ist.
`kubectl get cluster-api`	Listet die Cluster-API-Ressourcen auf, die die Cluster im aktuellen Namespace unterstützen, einschließlich der Ressourcen aus dem Cluster-API-Projekt und aus dem vom Tanzu Kubernetes Grid-Dienst verwendeten Cluster-API-Anbieter.
`kubectl get tanzukubernetesreleases`	Listen Sie die verfügbaren Tanzu Kubernetes-Versionen auf.
`kubectl get tkr`	Kurzformversion des vorhergehenden Befehls.
`kubectl get tkr` *`v1.23.8---vmware.1-tkg.1.5417466`* `-o yaml`	Stellt Details zur benannten Tanzu Kubernetes-Version bereit.
`kubectl get virtualmachine`	Listet die VM-Ressourcen auf, die die Clusterknoten im aktuellen Namespace unterstützen.
`kubectl get vm`	Kurzformversion des vorhergehenden Befehls.
`kubectl describe virtualmachine VIRTUAL-MACHINE-NAME`	Beschreibt die angegebene virtuelle Maschine und zeigt den Zustand, den aktuellen Status und die Ereignisse an.
`kubectl describe virtualmachinesetresourcepolicy`	Listet die VM-Ressourcen zum Festlegen einer Ressourcenrichtlinie auf, die den Cluster im aktuellen Namespace unterstützen. Diese Ressourcen stellen den Ressourcenpool für die vSphere-Objekte und den Ordner dar, die für den Cluster verwendet werden.
`kubectl get virtualmachineservice`	Listet die VM-Dienstressourcen auf, die die Clusterknoten im aktuellen Namespace unterstützen. Diese Ressourcen sind analog zu einem Dienst, allerdings nicht für Pods, sondern für virtuelle Maschinen. VM-Dienste werden sowohl für die Bereitstellung eines Load Balancers für die Knoten der Steuerungsebene eines Clusters als auch vom Anbieter paravirtueller Clouds verwendet, um in einem Cluster einen Kubernetes-Dienst vom Typ „LoadBalancer“ zu unterstützen.

Tabelle 8-11. Benutzerdefinierte Befehle zum Verwalten von TKG-Clustern (Fortsetzung)

Befehl	Beschreibung
`kubectl get vmservice`	Kurzformversion des vorhergehenden Befehls.
`kubectl describe virtualmachineservice` *`VIRTUAL-MACHINE-SERVICE-NAME`*	Beschreibt den angegebenen VM-Dienst und zeigt den angegebenen Clusterzustand, den aktuellen Status und die Ereignisse an.
`kubectl get virtualmachineimage`	Listet die verfügbaren VM-Images auf.
`kubectl get vmimage`	Kurzversion des vorhergehenden Befehls.
`kubectl describe vmimage VM_IMAGE_NAME`	Zeigen Sie Details zum benannten VM-Image an.
`kubectl get virtualnetwork`	Listet die virtuellen Netzwerkressourcen im aktuellen Namespace auf, einschließlich der für Cluster verwendeten Ressourcen. Für jeden Namespace, in dem ein Cluster bereitgestellt wird, und für die einzelnen Cluster selbst wird ein virtuelles Netzwerk erstellt.
`kubectl get persistentvolumeclaim`	Listet die Ressourcen für die Beanspruchung eines dauerhaften Volumes im aktuellen Namespace auf, einschließlich der für Cluster verwendeten Ressourcen.
`kubectl get cnsnodevmattachment`	Listet die Ressourcen für VM-Anhänge von CNS-Knoten im aktuellen Namespace auf. Diese Ressourcen stellen den Anhang eines von CNS verwalteten dauerhaften Volumes an eine virtuelle Maschine dar, die als Knoten eines Clusters dient.
`kubectl get configmap`	Listet die Konfigurationszuordnungen im aktuellen Namespace auf, einschließlich der für die Erstellung von Clusterknoten verwendeten Zuordnungen. Konfigurationszuordnungen können nicht von Benutzern geändert werden. Alle Änderungen werden überschrieben.
`kubectl get secret`	Listet die geheimen Schlüssel im aktuellen Namespace auf, einschließlich geheimer Schlüssel, die für die Erstellung und Verwaltung von Clusterknoten verwendet werden.

Anzeigen des Lebenszyklusstatus von TKG-Clustern

Sie können den Lebenszyklusstatus von TKG-Clustern in der vSphere-Bestandsliste sowie mithilfe von kubectl anzeigen.

Lebenszyklusstatus von TKG-Clustern in vSphere

In der Tabelle werden die Statusinformationen zum TKG-Cluster, die in der vSphere-Bestandsliste angezeigt werden, aufgelistet und beschrieben.

Tabelle 8-12. Status des TKG-Clusters in der vSphere-Bestandsliste

Bereich	Beschreibung	Beispiel
Name	Der benutzerdefinierte Name des Clusters.	`tkg2-cluster-01`
Erstellungszeit	Das Datum und die Uhrzeit der Clustererstellung.	`Mar 17, 2022, 11:42:46 PM`
Phase	Der Lebenszyklusstatus des Clusters.	`creating`
Worker-Anzahl	Die Anzahl der Worker-Knoten im Cluster.	`1` oder `2` oder `5`
Verteilungsversion	Die Version der Kubernetes-Software, die im Cluster ausgeführt wird.	`v1.22.6+vmware.1-tkg.1.7144628`
Adresse der Steuerungsebene	Die IP-Adresse des Load Balancers der Steuerungsebene des Clusters.	`192.168.123.2`

Lebenszyklusstatus von TKG-Clustern in kubectl

In der Tabelle werden die Statusinformationen zum TKG-Cluster, die in kubectl angezeigt werden, aufgelistet und beschrieben.

Tabelle 8-13. Status von TKG-Clustern in kubectl

Bereich	Beschreibung	Beispiel
NAME	Name des Clusters.	`tkg2-cluster-01`
CONTROL PLANE	Anzahl der Knoten der Steuerungsebene im Cluster.	`3`
WORKER	Anzahl der Worker-Knoten im Cluster.	`5`
DISTRIBUTION	Die Kubernetes-Version, auf der der Cluster ausgeführt wird.	`v1.22.6+vmware.1-tkg.1.5b5608b`
AGE	Anzahl der Tage, an denen der Cluster ausgeführt wurde.	`13d`
PHASE	Der Lebenszyklusstatus des Clusters.	`running`

Status der Lebenszyklusphase des Clusters

In der Tabelle wird der Status für jede Phase des Lebenszyklus eines Clusters aufgelistet und beschrieben.

Tabelle 8-14. Status der Lebenszyklusphase des Clusters

Phase	Beschreibung
`creating`	Die Clusterbereitstellung kann beginnen, die Steuerungsebene wird erstellt – oder sie wird erstellt, aber nicht initialisiert.
`deleting`	Der Cluster wird gelöscht.

Tabelle 8-14. Status der Lebenszyklusphase des Clusters (Fortsetzung)

Phase	Beschreibung
`failed`	Die Erstellung der Clustersteuerungsebene ist fehlgeschlagen, und wahrscheinlich ist ein Eingreifen des Benutzers erforderlich.
`running`	Die Infrastruktur wird erstellt und konfiguriert, und die Steuerungsebene wird vollständig initialisiert.
`updating`	Ein Update des Clusters wird durchgeführt.

Anzeigen der Ressourcenhierarchie für einen TKG-Cluster mithilfe von kubectl

Sie können die Ressourcenhierarchie für einen TKG-Cluster mithilfe von kubectl anzeigen. Wenn Sie die vollständige Liste der Clusterressourcen anzeigen, können Sie leichter Ressourcen identifizieren, die möglicherweise zu Problemen führen.

Verfahren

1 Stellen Sie eine Verbindung mit Supervisor her und melden Sie sich an.

2 Ändern Sie den Kontext in den vSphere-Namespace, in dem der TKG-Cluster bereitgestellt wird.

```
kubectl config use-context tkg2-cluster-ns
```

3 Starten Sie den folgenden Befehl, um die Clusterressource für Cluster-API anzuzeigen.

```
kubectl describe clusters.cluster.x-k8s.io CLUSTER-NAME
```

Dieser Befehl gibt die Ressourcenhierarchie für den benannten Cluster zurück, einschließlich Kubernetes-Namespace, API-Version und Ressourcenversion.

Konfigurieren von MachineHealthCheck für v1beta1-Cluster

In diesem Thema wird die Konfiguration von MachineHealthCheck für TKG-Dienst-Cluster beschrieben, die mithilfe der v1beta1-API bereitgestellt werden.

MachineHealthCheck für v1beta1-Cluster

Bei MachineHealthCheck handelt es sich um die API-Ressource eines Kubernetes-Clusters, die Bedingungen für die Standardisierung fehlerhafter Maschinen definiert. In Kubernetes stellt eine Maschine eine benutzerdefinierte Ressource dar, die kubelet ausführen kann. Auf der vSphere IaaS control plane wird eine Kubernetes-Maschinenressource durch eine vSphere-VM gestützt. Weitere Informationen finden Sie in der Upstream-Dokumentation.

Wenn Sie einen Cluster mithilfe des TKG-Dienst bereitstellen, werden vom System standardmäßige MachineHealthCheck-Objekte erstellt – eines für alle Steuerungsebenen und eines für jede Maschinenbereitstellung. Ab vSphere 8 Update 3 können Maschinenintegritätsprüfungen für v1beta1-Cluster konfiguriert werden. Zu den unterstützten Einstellungen gehören:

- maxUnhealthy
- nodeStartupTimeout
- unhealthyConditions
- unhealthyRange

In der Tabelle werden die unterstützten Vorgänge für die Maschinenintegritätsprüfungen beschrieben.

Tabelle 8-15. Maschinenintegritätsprüfungen

Bereich	Wert	Beschreibung
`maxUnhealthy`	string Absolute Zahl oder ein Prozentsatz	Die Standardisierung wird nicht durchgeführt, wenn die Anzahl der fehlerhaften Maschinen den Wert überschreitet.
`nodeStartupTimeout`	string Dauer im Format `XhXmXs` (Stunden, Minuten, Sekunden)	Jede zu erstellende Maschine, die beim Beitritt zum Cluster den angegebenen Zeitraum überschreitet, gilt als fehlgeschlagen und wird standardisiert.

Tabelle 8-15. Maschinenintegritätsprüfungen (Fortsetzung)

Bereich	Wert	Beschreibung
`unhealthyConditions`	Array [] von UnhealthyConditions-Typen Verfügbare Bedingungstypen: [`Ready`, `MemoryPressure`,`DiskPressure`, `PIDPressure`, `NetworkUnavailable`] Verfügbare Bedingungsstatuszustände: [`True`, `False`, `Unknown`]	Liste der Bedingungen, die bestimmen, ob ein Steuerungsebenenknoten als fehlerhaft gilt.
`unhealthyRange`	string	Eine weitere Standardisierung ist nur zulässig, wenn die Anzahl der von „Selektor“ als nicht fehlerfrei ausgewählten Maschinen innerhalb des Bereichs von `unhealthyRange` liegt. Hat Vorrang vor `maxUnhealthy`. Beispiel: „[3-5]“ bedeutet, dass die Standardisierung nur zulässig ist, wenn (a) mindestens 3 fehlerhafte Maschinen und (b) höchstens 5 fehlerhafte Maschinen vorhanden sind.

Hinweis MachineHealthCheck-Objekte werden für v1alpha3-Cluster bereitgestellt, können aber nicht konfiguriert werden. Details hierzu finden Sie unter Überprüfen der Integrität der TKG-Clustermaschine mithilfe von Kubectl.

MachineHealthCheck – Beispiel

Im folgenden Beispiel wird eine `machineHealthCheck` für eine bestimmte `machineDeployment` konfiguriert.

```
...
  topology:
    class: tanzukubernetescluster
    version: v1.28.8---vmware.1-fips.1-tkg.2
    controlPlane:
      machineHealthCheck:
        enable: true
        maxUnhealthy: 100%
        nodeStartupTimeout: 4h0m0s
        unhealthyConditions:
        - status: Unknown
          timeout: 5m0s
          type: Ready
       - status: "False"
          timeout: 12m0s
          type: Ready
      ...
    workers:
      machineDeployments:
```

```
    - class: node-pool
      failureDomain: np1
      machineHealthCheck:
        enable: true
        maxUnhealthy: 100%
        nodeStartupTimeout: 4h0m0s
        unhealthyConditions:
        - status: Unknown
          timeout: 5m0s
          type: Ready
        - status: "False"
          timeout: 12m0s
          type: Ready
```

Patchen von MachineHealthCheck mithilfe von Kubectl

Zum Aktualisieren der `MachineHealthCheck` für einen v1beta1-Cluster nach dessen Bereitstellung verwenden Sie die Methode `patch`.

Vorsicht Diese Anweisungen enthalten eine allgemeine Anleitung zum Patchen eines vorhandenen Clusters. Die verwendeten Werte richten sich nach Ihrer Umgebung und dem bereitgestellten Cluster, den Sie patchen möchten. Verwenden Sie die Tanzu-CLI, um die `MachineHealthCheck` für einen vorhandenen Cluster zu patchen.

1 Rufen Sie die `machineDeployment` aus der Definition der Clusterressource ab.

```
kubectl get cluster CLUSTER_NAME -o yaml
```

Im Abschnitt `spec.topology.workers.machineDeployments` sollte der jeweilige Wert für die entsprechende `machineDeployment` angezeigt werden.

2 Löschen Sie die MachineHealthCheck für den Worker-Knoten.

```
kubectl patch cluster <Cluster Name> -n <cluster namespace>  --type json -p='{"op":
"replace", "path": "/spec/topology/workers/machineDeployments/<index>/machineHealthCheck",
"value":{"enable":false}}'
```

3 Löschen Sie die MachineHealthCheck für die Steuerungsebene.

```
kubectl patch cluster <cluster-name> -n <cluster-namespace> --type json
-p='{"op": "replace", "path": "/spec/topology/controlPlane/machineHealthCheck", "value":
{"enable":false}}'
```

4 Erstellen oder aktualisieren Sie die MachineHealthCheck für die Steuerungsebene mit den gewünschten Einstellungen.

```
kubectl patch cluster <cluster-name> -n <cluster-namespace> --type json
-p='[{"op": "replace", "path": "/spec/topology/controlPlane/machineHealthCheck",
"value":{"enable":true,"nodeStartupTimeout":"1h58m","unhealthyConditions":
[{"status":"Unknown","timeout":"5m10s","type":"Unknown"},
{"status":"Unknown","timeout":"5m0s","type":"Ready"}],"maxUnhealthy":"100%"}}]'
```

5 Erstellen oder aktualisieren Sie die MachineHealthCheck für den Worker-Knoten mit den gewünschten Einstellungen.

```
kubectl patch cluster <cluster-name> -n <cluster-namespace> --type json -p='[{"op":
"replace", "path": "/spec/topology/workers/machineDeployments/<index>/machineHealthCheck",
"value":{"enable":true,"nodeStartupTimeout":"1h58m","unhealthyConditions":
[{"status":"Unknown","timeout":"5m10s","type":"Unknown"},
{"status":"Unknown","timeout":"5m0s","type":"Ready"}],"maxUnhealthy":"100%"}}]'
```

Konfigurieren von MachineHealthCheck mithilfe der Tanzu-CLI

Mithilfe der Tanzu-CLI können Sie MachineHealthCheck für einen v1beta1-Cluster erstellen.

Führen Sie beispielsweise folgenden Befehl aus, um die MachineHealthCheck-Einstellungen für die Steuerungsebene zu erstellen oder zu aktualisieren.

```
tanzu cluster mhc control-plane set <cluster-name> --node-startup-timeout 2h7m10s
```

Führen Sie folgenden Befehl aus, um sicherzustellen, dass die Einstellung aktualisiert und nicht abgeglichen wird.

```
tanzu cluster mhc control-plane get <cluster-name>
```

Führen Sie folgenden Befehl aus, um die MachineHealthCheck-Einstellungen für die Maschinenbereitstellung zu erstellen oder zu aktualisieren.

```
tanzu cluster mhc node set <cluster-name> --machine-deployment node-pool-1 --node-startup-
timeout 1h59m0s
```

Führen Sie folgenden Befehl aus, um sicherzustellen, dass die Einstellung aktualisiert und nicht abgeglichen wird.

```
tanzu cluster mhc node get <cluster-name> -m <cluster-name>-node-pool-1-nr7r5
```

Neben „get“ und „set“ wird vom System auch der Löschvorgang unterstützt. Beispiel:

Für die Steuerungsebene können Sie beispielsweise folgenden Befehl verwenden:

```
tanzu cluster mhc control-plane delete <cluster-name>
```

Für den Knoten können Sie beispielsweise folgenden Befehl verwenden:

```
tanzu cluster mhc <cluster-name> --machine-deployment <machine deployment name>
```

Aktualisieren von TKG-Dienstclustern

9

Dieser Abschnitt enthält eine Anleitung für die Aktualisierung von TKG-Dienstclustern.

Lesen Sie als Nächstes die folgenden Themen:

- Grundlegendes zum Modell für parallele Updates für TKG-Dienstcluster
- Überprüfen der TKGS-Clusterkompatibilität für Updates
- Aktualisieren eines TKG-Clusters durch Bearbeiten der TKR-Version
- Aktualisieren eines TKG-Clusters durch Bearbeiten der Speicherklasse
- Aktualisieren eines TKG-Dienstclusters durch Bearbeiten der VM-Klasse
- Aktualisieren eines TKG-Clusters mithilfe der Tanzu-CLI

Grundlegendes zum Modell für parallele Updates für TKG-Dienstcluster

TKG-Dienstcluster unterstützen ein Modell für parallele Updates. Sie können ein paralleles Update initiieren, indem Sie die Clusterspezifikation ändern. Einige Systemvorgänge können ein paralleles Update initialisiert haben. Bevor Sie die Umgebung aktualisieren, sollten Sie sich mit dem Vorgang des parallelen Updates vertraut machen.

Modell für parallele Updates für TKGS-Cluster nach TKG-Dienst 3.0

Ab TKG-Dienst 3.0 ist der TKG-Controller unabhängig von vCenter Server und vom Supervisor. Weitere Informationen hierzu finden Sie unter Verwenden der TKG-Dienst. Durch das Upgrade dieser Komponenten wird kein paralleles Update von TKGS-Clustern aktiviert.

Das Upgrade der TKG-Dienstversion kann ein paralleles Update von TKGS-Clustern auslösen.

Modell für parallele Updates für TKGS-Cluster vor TKG-Dienst 3.0

Der TKG-Controller wird auf Supervisor ausgeführt. Wenn Sie Supervisor aktualisieren, wird der TKG-Controller automatisch aktualisiert, wenn ein Update verfügbar ist. Jedes TKG-Controller-Update kann Aktualisierungen für unterstützende Dienste wie z. B. CNI, CSI, CPI sowie Konfigurationsaktualisierungen für Cluster enthalten. Zum Zweck der Kompatibilität führt das System Vorabprüfungen durch und erzwingt die Konformität.

vSphere IaaS control plane unterstützt ein Modell für parallele Updates für TKG-Cluster auf Supervisor. Das Modell für parallele Updates minimiert die Ausfallzeiten während der Aktualisierung. Parallele Updates umfassen Upgrades der Kubernetes-Versionen sowie der Infrastruktur und der Dienste, die die Cluster unterstützen, wie z. B. Konfigurationen und Ressourcen virtueller Maschinen, Dienste und Namespaces sowie benutzerdefinierte Ressourcen. Damit das Update erfolgreich verläuft, muss Ihre Konfiguration verschiedene Kompatibilitätsanforderungen erfüllen, sodass das System Bedingungen für die erneute Prüfung erzwingt, um sicherzustellen, dass die Cluster für das Updates bereit sind, und Rollback unterstützt, falls das Cluster-Upgrade nicht erfolgreich ist.

Sie können ein paralleles Update eines TKG-Clusters initiieren, indem Sie d en Wert des Parameters in der Clusterspezifikation ändern. Ein paralleles Cluster-Update kann auch vom System initiiert werden. Wird beispielsweise ein vSphere-Namespaces-Update durchgeführt, gibt das System die aktualisierten Konfigurationen sofort an alle Arbeitslastcluster weiter. Diese Aktualisierungen können ein paralleles Update von Clusterknoten auslösen. Zudem kann eine Änderung an einem der Konfigurationselemente auch zur Initiierung eines parallelen Updates führen. Wenn beispielsweise das einer Verteilungsversion entsprechende `VirtualMachineImage` umbenannt oder ersetzt wird, wird ein paralleles Update initiiert, da das System versucht, alle auf dem neuen Image ausgeführten Knoten abzurufen. Darüber hinaus löst das Aktualisieren eines Supervisors wahrscheinlich ein paralleles Update der dort bereitgestellten Arbeitslast-Cluster aus. Wenn beispielsweise der `vmware-system-tkg-controller-manager` aktualisiert wird, fügt das System neue Werte in den Manifestgenerator ein, und der Controller initiiert ein paralleles Update zur Bereitstellung dieser Werte.

Der Vorgang des parallelen Updates zum Ersetzen der Clusterknoten ähnelt dem parallelen Update von Pods in einer Kubernetes-Bereitstellung. Es gibt zwei verschiedene Controller, die für das Durchführen eines parallelen Updates von Arbeitslast-Clustern verantwortlich sind: den Add-On-Controller und den Cluster-Controller. Innerhalb dieser beiden Controller gibt es drei wichtige Phasen für ein paralleles Update: das Aktualisieren von Add-Ons, das Aktualisieren der Steuerungsebene und das Aktualisieren der Worker-Knoten. Diese Phasen erfolgen in der genannten Reihenfolge mit Vorabprüfungen, die verhindern, dass ein Schritt gestartet wird, bevor der vorherige Schritt ausreichend fortgeschritten ist. Diese Schritte werden möglicherweise übersprungen, wenn sie als unnötig eingestuft wurden. Ein Update kann beispielsweise nur Worker-Knoten betreffen und daher keine Add-On- oder Steuerungsebenen-Updates erfordern.

Während des Updatevorgangs fügt das System einen neuen Clusterknoten hinzu und wartet darauf, dass der Knoten mit der Kubernetes-Zielversion in den Onlinemodus wechselt. Das System markiert dann den alten Knoten zum Löschen, wechselt zum nächsten Knoten und wiederholt den Vorgang. Der alte Knoten wird erst gelöscht, wenn alle Pods entfernt wurden. Wenn ein Pod beispielsweise mit PodDisruptionBudgets definiert ist, die verhindern, dass ein Knoten vollständig entleert wird, wird der Knoten isoliert, aber erst gelöscht, wenn diese Pods entfernt werden können. Das System aktualisiert zuerst alle Knoten der Steuerungsebene und dann die Worker-Knoten. Während eines Updates ändert sich der Status des Clusters in „wird aktualisiert“. Nach Abschluss des parallelen Updates ändert sich der Status des Clusters in „wird ausgeführt“.

Auf einem Cluster ausgeführte Pods, die nicht von einem Replizierungs-Controller gesteuert werden, werden während des Upgrades einer Kubernetes-Version, das im Rahmen der Entleerung des Worker-Knotens beim Update des Clusters stattfindet, gelöscht. Dies ist der Fall, wenn das Cluster-Update manuell oder automatisch durch ein vSphere Namespaces- oder Supervisor-Update ausgelöst wird. Zu den nicht durch einen Replizierungs-Controller gesteuerten Pods zählen Pods, die nicht als Teil einer Bereitstellungs- oder ReplicaSet-Spezifikation erstellt werden. Weitere Informationen finden Sie in der Kubernetes-Dokumentation unter Pod Lifecycle: Pod lifetime (Pod-Lebenszyklus: Pod-Lebensdauer).

Vom Benutzer initiierte parallele Updates

Sie können ein paralleles Update eines TKG-Clusters auf Supervisor initiieren, indem Sie die Tanzu Kubernetes-Version-Version, die Klasse der virtuellen Maschine und die Speicherklasse aktualisieren. Einzelheiten finden Sie in einem der folgenden Themen.

- Aktualisieren eines TKG-Clusters durch Bearbeiten der TKR-Version
- Aktualisieren eines TKG-Clusters durch Bearbeiten der Speicherklasse
- Aktualisieren eines TKG-Dienstclusters durch Bearbeiten der VM-Klasse
- Aktualisieren eines TKG-Clusters mithilfe der Tanzu-CLI

Vom System initiierte parallele Updates

In jeder Version von Supervisor können Änderungen an einem oder mehreren der folgenden Objekte vorgenommen werden:

- kubeadmcontrolplanetemplate/kubeadmcontrolplane
- kubeadmconfigtemplate/kubeadmconfig
- vpheremachinetemplate/vspheremachine (für vSphere 8.x)
- vcpmachinetemplate/wcpmachine (für vSphere 7.x)

Wenn Supervisor aktualisiert wird, lösen die zentralen CAPI-Controller (Cluster API) ein Update-Rollout für TKG-Arbeitslastcluster aus, um den gewünschten Zustand der oben genannten Objekte mit den ausgeführten Arbeitslastclustern abzugleichen.

In vSphere IaaS control plane erzeugt der in Supervisor ausgeführte TKG-Controller diese Objekte und synchronisiert sie fortlaufend mit dem Systemcode. Wenn die Controller folglich auf einen neueren Code aktualisiert werden, führen Änderungen an einem der oben genannten Objekte zu einem parallelen Update der vorhandenen TKG-Cluster. Änderungen am Systemcode, die sich auf Supervisor auswirken, führen folglich zu parallelen Updates von TKG-Clustern.

In der Tabelle werden die Bedingungen beschrieben, unter denen Sie bei einem Upgrade von Supervisor ein automatisiertes paralleles Update von Arbeitslastclustern erwarten können.

Upgrade-Szenario	Beschreibung
Upgrade von einer beliebigen vCenter Server 7.x-Version auf eine beliebige vCenter Server-Version	Löst möglicherweise ein paralleles Update aller Tanzu Kubernetes-Cluster aus. Ein paralleles Update wird durch das erste Upgrade vom Supervisor nach einem Upgrade von vCenter Server ausgelöst. Ein paralleles Update wird in der Regel nicht durch ein Upgrade vom Supervisor auf demselben vCenter Server ausgelöst. Spezifische Details finden Sie in den Versionshinweisen.
Upgrade von einer beliebigen vCenter Server-Version auf eine beliebige vCenter Server 8.x-Version	Löst ein paralleles Update aller TKG-Cluster aus, da die folgenden Codeänderungen weitergegeben werden müssen: ■ Zugrunde liegende CAPI-Anbieter müssen von CAPW zu CAPV verschoben werden ■ Migrieren der Cluster von klassenlosen CAPI-Clustern zu einem CAPI-Cluster mit Klassen
Upgrade von vCenter Server Version 8.0 GA (8.0.0) auf die Versionen vCenter Server 8.0.0b oder 8.0.0c	Löst ein paralleles Update der angegebenen TKG-Cluster aus, wenn einer der folgenden Fälle zutrifft: ■ Jeder TKG-Cluster, der Proxy-Einstellungen mit einer nicht leeren noProxy-Liste verwendet hat. ■ Alle TKG-Cluster, wenn der eingebettete Habor-Registrierungsdienst auf dem Supervisor aktiviert war.
Upgrade von vSphere Version 8.0.0b auf vSphere Version 8.0.0c	Keine automatischen Rollouts von Arbeitslastclustern
Upgrade von vSphere Version 8.0.0c auf vSphere 8.0 Version Update 1 (8.0.1)	Keine automatischen Rollouts von Arbeitslastclustern
Upgrade von einer beliebigen vSphere 8.x-Version auf eine 8.0 U2-Version (8.0.2)	Dies führt zu einem parallelen Upgrade für alle TKCs, da die folgenden Änderungen vorgenommen werden müssen: ■ vSphere 8.0 U2 umfasst STIG-Änderungen auf Kubernetes-Ebene für TKG 1.0- und TKG 2.0-TKRs in GCM als Teil der ClusterClass. ■ Da TKCs ab Version 1.23 mit 8.0 U2 kompatibel sind, muss für alle Cluster ein paralleles Upgrade durchgeführt werden.
Upgrade von einer beliebigen vSphere 8.x-Version vor 8.0 U2 (8.0.2) auf Version 8.0 U2c	Dies führt zu einem parallelen Upgrade für alle TKCs, da die folgenden Änderungen vorgenommen werden müssen: ■ 8.0U2 umfasst STIG-Änderungen auf der k8s-Ebene für TKG 1.0- und TKG 2.0-TKRs in GCM als Teil der ClusterClass. ■ Da TKCs ab Version 1.23 mit 8.0 P03 kompatibel sind, muss für alle Cluster ein paralleles Upgrade durchgeführt werden.

Darüber hinaus kann das Ändern der Inhaltsbibliothek, in der TKR-Images gehostet werden, parallele Updates von TKG-Clustern auslösen. Das Hinzufügen neuer Images über ein Abonnement oder manuell löst kein paralleles Update von TKG-Clustern aus. Das Ändern der Inhaltsbibliothek und das Hinzufügen von Images mit unterschiedlichen Namen hingegen löst ein paralleles Update aller TKG-Cluster aus.

Betrachten Sie beispielsweise ein Szenario, in dem Sie eine abonnierte Inhaltsbibliothek verwenden, die automatisch die systemdefinierten OVA-Namen verwendet. Wechseln Sie anschließend zu einer lokalen Inhaltsbibliothek und befüllen Sie sie mit denselben OVAs. Verwenden Sie jedoch andere Namen. Auf diese Weise wird ein paralleles Update aller TKG-Cluster ausgelöst, da die Ersatzinhaltsbibliothek dieselben OVAs, aber unterschiedliche benutzerdefinierte Namen aufweist.

Überlegungen zu parallelen Updates für Cluster mit mehreren Knotenpools

Wenn Sie TKG-Cluster mit mehreren Knotenpools verwenden, beachten Sie die folgenden Informationen in Bezug auf parallele Updates.

Worker-Knotenpools

Worker-Knotenpools wurden mit der TKGS-v1alpha2-API eingeführt, die mit vSphere 7 U3 veröffentlicht wurde. Cluster API MachineDeployments ist das zugrunde liegende Kubernetes-Primitiv von Worker-Knotenpools.

ClusterClass wurde mit der vSphere 8-Version von TKGS eingeführt. Sowohl die v1alpha3- als auch die v1beta1-APIs basieren auf ClusterClass. (v1alpha3 ist eine Abstraktionsschicht auf ClusterClass.)

Aktualisieren mehrerer Knotenpools während des parallelen Updates

Wenn Sie einen TKGS-Arbeitslastcluster aktualisieren, der mit mehreren Knotenpools bereitgestellt wird, unterscheidet sich das Modell für parallele Updates je nach verwendeter Version von vSphere.

vSphere	TKGS-API	Upgrade-Verhalten
vSphere 7 TKGS	v1alpha2-API	Mehrere Knotenpools innerhalb desselben Clusters werden zur selben Zeit (gleichzeitig) aktualisiert
vSphere 8 TKGS	v1alpha3-API und v1beta1-API	Mehrere Knotenpools innerhalb desselben Clusters werden nach einer logischen Reihenfolge (sequenziell) aktualisiert.

Best Practice-Überlegungen

Die Bereitstellung eines vSphere 8-TKGS-Clusters mit mehreren identischen Knotenpools ist im Hinblick auf die Dimensionierung irrelevant. Knotenpools sollten für verschiedene Größen, VM-Klassen, TKr-Versionen usw. verwendet werden. Vermeiden Sie es, mehrere identische Knotenpools zu verwenden, um das System herauszufordern und Cluster schneller zu aktualisieren, da dies nicht funktioniert.

Mit Pod Disruption Budgets können Sie sicherstellen, dass Upgrades ausgeführte Anwendungen nicht beeinträchtigen. Dazu empfiehlt es sich, PodDisruptionBudgets für die Arbeitslasten festzulegen (siehe https://kubernetes.io/docs/tasks/run-application/configure-pdb/). Die Cluster-API berücksichtigt diese und beendet eine Maschine nicht, wenn die Schwellenwerte überschritten würden.

Details zu parallelen Updates für vSphere 8 TKGS-Cluster

Während einer Aktualisierung der TKGS-Clusterversion auf vSphere 8:

- Die Knoten der Steuerungsebene werden zuerst aktualisiert. Anschließend werden Worker-Knoten beginnend mit dem Knotenpool „Zone-A“ nacheinander bereitgestellt. Wenn zwei Knotenpools verwendet werden, wird jeweils nur ein Worker bereitgestellt.

Bei Aktualisierungen der Clusterkonfigurationsvariablen:

- Die Knoten der Steuerungsebene werden zuerst aktualisiert, dann wird ein Worker-Knoten pro Knotenpool bereitgestellt. Wenn beispielsweise zwei Knotenpools verwendet werden, werden 2 Worker gleichzeitig bereitgestellt.

Überprüfen der TKGS-Clusterkompatibilität für Updates

Vor dem Upgrade eines TKGS-Arbeitslastclusters sollten Sie seine Kompatibilität mit dem Upgrade überprüfen. Die Kompatibilität muss mit dem TKG-Dienst abgeglichen werden.

Überprüfen der Kompatibilität mit dem TKG-Dienst

Vor dem Upgrade eines Arbeitslastclusters sollten Sie seine Kompatibilität mit dem Upgrade überprüfen. Wenn ein Cluster nicht mit dem TKG-Dienst kompatibel ist, führen Sie ein Upgrade der Tanzu Kubernetes-Version durch. Weitere Informationen zu verfügbaren TKrs finden Sie in den Versionshinweisen. Weitere Informationen finden Sie auch in der Online-Interoperabilitätsmatrix.

Sie können Tanzu Kubernetes-Versionen mit dem folgenden Befehl auflisten und deren Kompatibilität anzeigen.

```
kubectl get tkr
```

Die Spalte `COMPATIBLE` zeigt an, ob diese Tanzu Kubernetes-Version mit dem aktuellen TKG-Dienst installiert sind. Ab TKG-Dienst Version 3.1 gibt die Spalte TYPE auch den Kompatibilitätsstatus zurück.

Wenn Sie den TKGS-Cluster angeben, können Sie anzeigen, welche TKr-Updates verfügbar sind.

Mit der v1alpha3-API:

```
kubectl get tkc <tkgs-cluster-name>
```

Oder mit der v1beta1-API:

```
kubectl get cc <tkgs-cluster-name>
```

Die Spalte `UPDATES AVAILABLE` gibt an, ob ein Kubernetes-Upgrade verfügbar ist und zeigt die nächsten zu verwendenden Tanzu Kubernetes-Versionen-Empfehlungen an. Beispiel:

```
kubectl get tkc tkg2-cluster-11-tkc
NAME                   CONTROL PLANE   WORKER   TKR NAME                           AGE
READY   TKR COMPATIBLE   UPDATES AVAILABLE
tkg2-cluster-11-tkc    3               3        v1.25.7---vmware.3-fips.1-tkg.1    13d
True    True             [v1.26.5+vmware.2-fips.1-tkg.1]
```

Es gibt zwei Arten von TKr-Formaten: Nicht-Legacy und Legacy.

- Nicht-Legacy-TKrs wurden speziell für vSphere 8.x entwickelt und sind nur mit vSphere 8.x kompatibel.
- Legacy-TKrs verwenden ein Legacy-Format, das mit vSphere 7.x und auch mit vSphere 8.x kompatibel ist, aber nur zu Upgrade-Zwecken.

So listen Sie Nicht-Legacy-TKrs auf:

```
kubectl get -l !run.tanzu.vmware.com/legacy-tkr
```

So listen Sie Legacy-TKrs auf:

```
kubectl get -l !run.tanzu.vmware.com/legacy-tkr
```

Aktualisieren eines TKG-Clusters durch Bearbeiten der TKR-Version

In dieser Aufgabe wird beschrieben, wie Sie die Tanzu Kubernetes-Version-Version für einen TKG-Cluster aktualisieren, indem Sie das TKG-Clustermanifest bearbeiten.

Sie können ein paralleles Update eines TKGS-Clusters initiieren, indem Sie mithilfe des Befehls `kubectl edit` ein Upgrade der Tanzu Kubernetes-Version-Version durchführen.

Hinweis Sie können den Befehl `kubectl apply` nicht verwenden, um die TKR-Version für einen bereitgestellten Cluster zu aktualisieren.

Voraussetzungen

Für diese Aufgabe muss der kubectl-Befehl edit verwendet werden. Mit diesem Befehl wird das Cluster-Manifest in dem durch Ihre KUBE_EDITOR- oder EDITOR-Umgebungsvariable definierten Texteditor geöffnet. Wenn Sie die Datei speichern, wird der Cluster mit den Änderungen aktualisiert. Informationen zum Konfigurieren eines Editors für kubectl zum Ausführen des Befehls `kubectl edit` finden Sie unter Konfigurieren eines Texteditors für Kubectl.

Verfahren

1 Authentifizieren Sie sich beim Supervisor.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 Ändern Sie den Kontext in den vSphere-Namespace, in dem der Arbeitslast-Zielcluster bereitgestellt wird.

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 Rufen Sie den TKG-Cluster und die Version ab.

v1alpha3-Cluster:

```
kubectl get tanzukubernetescluster
```

v1beta1-Cluster:

```
kubectl get cluster
```

4 Listen Sie die verfügbaren Tanzu Kubernetes-Versionen auf.

```
kubectl get tanzukubernetesreleases
```

5 Führen Sie den folgenden Befehl aus, um das Cluster-Manifest zu bearbeiten.

v1alpha3-Cluster:

```
kubectl edit tanzukubernetescluster/CLUSTER-NAME
```

v1beta1-Cluster:

```
kubectl edit cluster/CLUSTER-NAME
```

6 Bearbeiten Sie das Manifest, indem Sie die Tanzu Kubernetes-Version-Zeichenfolge aktualisieren.

Führen Sie beispielsweise für einen v1alpha3-Cluster eine Änderung von TKR v1.25.7:

```
topology:
  controlPlane:
   replicas: 1
   storageClass: vsan-default-storage-policy
   tkr:
    reference:
     name: v1.25.7---vmware.3-fips.1-tkg.1
   vmClass: guaranteed-large
  nodePools:
  - name: worker-tkg-pool01
   replicas: 3
   storageClass: vsan-default-storage-policy
   tkr:
    reference:
```

```
      name: v1.25.7---vmware.3-fips.1-tkg.1
    vmClass: guaranteed-large
    volumes:
    - capacity:
      storage: 128Gi
     mountPath: /var/lib/containerd
     name: containerd
```

Auf TKR v1.26.5:

```
topology:
  controlPlane:
   replicas: 1
   storageClass: vsan-default-storage-policy
   tkr:
    reference:
     name: v1.26.5---vmware.2-fips.1-tkg.1
   vmClass: guaranteed-large
  nodePools:
  - name: worker-tkg-pool01
   replicas: 3
   storageClass: vsan-default-storage-policy
   tkr:
    reference:
     name: v1.26.5---vmware.2-fips.1-tkg.1
   vmClass: guaranteed-large
   volumes:
   - capacity:
     storage: 128Gi
    mountPath: /var/lib/containerd
    name: containerd
```

Hinweis Die Steuerungsebenen- und Worker-Knoten müssen dieselbe TKR-Version aufweisen. Sie können alle TKr-Instanzen aktualisieren oder die Version der Steuerungsebene aktualisieren und den TKR-Namen aus den Worker-Knoten entfernen.

Führen Sie beispielsweise für einen v1beta1-Cluster eine Änderung von TKR v1.25.7:

```
apiVersion: cluster.x-k8s.io/v1beta1
  ...
  topology:
    class: tanzukubernetescluster
    version: v1.25.7---vmware.3-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      ...
    variables:
      ...
```

Auf TKR v1.26.5:

```
apiVersion: cluster.x-k8s.io/v1beta1
  ...
  topology:
    class: tanzukubernetescluster
    version: v1.26.5---vmware.2-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      ...
    variables:
      ...
```

7 Speichern Sie die Änderungen an der Manifestdatei.

Wenn Sie die Datei speichern, wendet kubectl die Änderungen auf den Cluster an. Im Hintergrund stellt der VM-Dienst auf dem Supervisor den neuen Worker-Knoten bereit.

8 Vergewissern Sie sich, dass kubectl berichtet, dass die Manifest-Bearbeitungen erfolgreich aufgezeichnet wurden.

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkg-cluster-1 edited
```

Hinweis Wenn Sie einen Fehler erhalten oder kubectl nicht meldet, dass das Cluster-Manifest erfolgreich bearbeitet wurde, sollten Sie überprüfen, ob Sie Ihren Standardtexteditor mithilfe der Umgebungsvariable KUBE_EDITOR richtig konfiguriert haben. Weitere Informationen hierzu finden Sie unter Konfigurieren eines Texteditors für Kubectl.

9 Überprüfen Sie, ob der Cluster aktualisiert wird.

```
kubectl get tanzukubernetescluster
NAME             CONTROL PLANE   WORKER   DISTRIBUTION                      AGE   PHASE
tkgs-cluster-1   3               3        v1.26.5---vmware.2-fips.1-tkg.1   21h   updating
```

10 Überprüfen Sie, ob der Cluster aktualisiert wurde.

```
kubectl get tanzukubernetescluster
NAME             CONTROL PLANE   WORKER   DISTRIBUTION                      AGE   PHASE
tkgs-cluster-1   3               3        v1.26.5---vmware.2-fips.1-tkg.1   22h   running
```

Aktualisieren eines TKG-Clusters durch Bearbeiten der Speicherklasse

Sie können einen TKG-Cluster aktualisieren, indem Sie die von den Clusterknoten verwendete Speicherklasse ändern.

Sie können ein paralleles Update eines TKG-Clusters initiieren. Bearbeiten Sie dazu mithilfe des Befehls `kubectl edit` den Wert des Parameters `storageClass` in der Clusterspezifikation.

Hinweis Sie können den Befehl `kubectl apply` nicht verwenden, um einen bereitgestellten TKG-Cluster zu aktualisieren.

Voraussetzungen

Für diese Aufgabe muss der kubectl-Befehl edit verwendet werden. Mit diesem Befehl wird das Cluster-Manifest in dem durch Ihre KUBE_EDITOR- oder EDITOR-Umgebungsvariable definierten Texteditor geöffnet. Wenn Sie die Datei speichern, wird der Cluster mit den Änderungen aktualisiert. Informationen zum Konfigurieren eines Editors für kubectl finden Sie unter Konfigurieren eines Texteditors für Kubectl.

Verfahren

1 Authentifizieren Sie sich beim Supervisor.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 Ändern Sie den Kontext in den vSphere-Namespace, in dem der Arbeitslast-Zielcluster bereitgestellt wird.

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 Um verfügbare Speicherklassen zu ermitteln und zu entscheiden, welche verwendet werden, führen Sie den folgenden Befehl aus.

```
kubectl describe tanzukubernetescluster CLUSTER-NAME
```

4 Führen Sie den folgenden Befehl aus, um das Cluster-Manifest zu bearbeiten.

v1alpha3-Cluster:

```
kubectl edit tanzukubernetescluster/CLUSTER-NAME
```

v1beta1-Cluster:

```
kubectl edit cluster/CLUSTER-NAME
```

5 Bearbeiten Sie das Manifest, indem Sie den `storageClass`-Wert ändern.

Ändern Sie beispielsweise für einen v1alpaha3-Cluster das Cluster-Manifest von der `silver-storage-class`-Klasse für Steuerungsebenen- und Worker-Knoten:

```
spec:
  topology:
    controlPlane:
      ...
```

```
      storageClass: silver-storage-class
    workers:
      ...
      storageClass: silver-storage-class
```

In die `gold-storage-class`-Klasse für Steuerungsebenen- und Worker-Knoten:

```
spec:
  topology:
    controlPlane:
      ...
      storageClass: gold-storage-class
    workers:
      ...
      storageClass: gold-storage-class
```

Wenn Sie einen v1beta1-Cluster bereitgestellt haben, aktualisieren Sie ebenso den `variables.storageclass`-Wert in der Clusterspezifikation mit dem Namen der Speicherklasse.

6 Speichern Sie die Änderungen an der Manifestdatei.

Wenn Sie die Datei speichern, wendet kubectl die Änderungen auf den Cluster an. Im Hintergrund stellt der Tanzu Kubernetes Grid die neuen Knoten-VMs bereit und fährt die alten herunter.

7 Vergewissern Sie sich, dass kubectl berichtet, dass die Manifest-Bearbeitungen erfolgreich aufgezeichnet wurden.

```
kubectl edit tanzukubernetescluster/tkgs-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkgs-cluster-1 edited
```

Hinweis Wenn Sie einen Fehler erhalten oder kubectl nicht meldet, dass das Cluster-Manifest erfolgreich bearbeitet wurde, sollten Sie überprüfen, ob Sie Ihren Standardtexteditor mithilfe der Umgebungsvariable KUBE_EDITOR richtig konfiguriert haben. Weitere Informationen hierzu finden Sie unter Konfigurieren eines Texteditors für Kubectl.

8 Überprüfen Sie, ob der Cluster aktualisiert wurde.

v1alpha3-Cluster:

```
kubectl get tanzukubernetescluster
```

v1beta1-Cluster:

```
kubectl get cluster
```

Aktualisieren eines TKG-Dienstclusters durch Bearbeiten der VM-Klasse

Sie können einen TKG-Dienstcluster aktualisieren, indem Sie die Klasse der virtuellen Maschine ändern, die zum Hosten der Clusterknoten verwendet wird.

Sie können ein paralleles Update eines TKG-Clusters initiieren. Bearbeiten Sie dazu mithilfe des Befehls `kubectl edit` die Definition `vmClass`. Neue Knoten basierend auf der geänderten Klasse werden bereitgestellt, und die alten Knoten werden heruntergefahren.

Hinweis Sie können den Befehl `kubectl apply` nicht verwenden, um einen bereitgestellten TKG-Cluster zu aktualisieren.

Voraussetzungen

Für diese Aufgabe muss der kubectl-Befehl edit verwendet werden. Mit diesem Befehl wird das Cluster-Manifest in dem durch Ihre KUBE_EDITOR- oder EDITOR-Umgebungsvariable definierten Texteditor geöffnet. Wenn Sie die Datei speichern, wird der Cluster mit den Änderungen aktualisiert. Informationen zum Konfigurieren eines Editors für kubectl finden Sie unter Konfigurieren eines Texteditors für Kubectl.

Verfahren

1 Authentifizieren Sie sich beim Supervisor.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 Ändern Sie den Kontext in den vSphere-Namespace, in dem der TKG-Zielcluster bereitgestellt wird.

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 Beschreiben Sie den TKG-Zielcluster und überprüfen Sie die VM-Klasse.

v1alpha3-Cluster:

```
kubectl describe tanzukubernetescluster CLUSTER-NAME
```

v1beta1-Cluster:

```
kubectl describe cluster CLUSTER-NAME
```

4 Listen Sie die verfügbaren VM-Klassen im vSphere-Namespace auf, in dem der Cluster bereitgestellt wird, und beschreiben Sie sie.

```
kubectl get virtualmachineclass
```

Hinweis Die VM-Zielklasse muss dem vSphere-Namespace zugeordnet werden, in dem der TKG-Cluster bereitgestellt ist. Weitere Informationen zum Binden von VM-Klassen an vSphere-Namespaces finden Sie in der Dokumentation zum TKG- oder VM-Dienst.

5 Führen Sie den folgenden Befehl aus, um das Cluster-Manifest zu bearbeiten.

v1alpha3-Cluster:

```
kubectl edit tanzukubernetescluster/CLUSTER-NAME
```

v1beta1-Cluster:

```
kubectl edit cluster/CLUSTER-NAME
```

6 Bearbeiten Sie das Manifest, indem Sie die Zeichenfolge der VM-Klasse ändern.

Wenn Sie beispielsweise einen v1alpah3-Cluster verwenden, ändern Sie das Cluster-Manifest von der Verwendung der VM-Klasse `guaranteed-medium` für Worker-Knoten:

```
topology:
   controlPlane:
     replicas: 3
     storageClass: vwk-storage-policy
     tkr:
       reference:
         name: v1.27.11---vmware.1-fips.1-tkg.2
     vmClass: guaranteed-medium
   nodePools:
   - name: worker-nodepool-a1
     replicas: 3
     storageClass: vwk-storage-policy
     tkr:
       reference:
         name: v1.27.11---vmware.1-fips.1-tkg.2
     vmClass: guaranteed-medium
```

So verwenden Sie die VM-Klasse `guaranteed-large` für Worker-Knoten:

```
topology:
   controlPlane:
     replicas: 3
     storageClass: vwk-storage-policy
     tkr:
       reference:
         name: v1.27.11---vmware.1-fips.1-tkg.2
     vmClass: guaranteed-medium
```

```
    nodePools:
    - name: worker-nodepool-a1
      replicas: 3
      storageClass: vwk-storage-policy
      tkr:
        reference:
          name: v1.27.11---vmware.1-fips.1-tkg.2
      vmClass: guaranteed-large
```

Wenn Sie einen v1beta1-Cluster bereitgestellt haben, aktualisieren Sie entsprechend den Wert von `variables.vmclass` auf die Ziel-VM-Klasse.

7 Speichern Sie die Änderungen an der Manifestdatei.

Wenn Sie die Datei speichern, wendet kubectl die Änderungen auf den Cluster an. Im Hintergrund stellt der TKG-Controller die neuen Knoten-VMs bereit und fährt die alten herunter.

8 Vergewissern Sie sich, dass kubectl berichtet, dass die Manifest-Bearbeitungen erfolgreich aufgezeichnet wurden.

```
kubectl edit tanzukubernetescluster/tkgs-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkgs-cluster-1 edited
```

Hinweis Wenn Sie einen Fehler erhalten oder kubectl nicht meldet, dass das Cluster-Manifest erfolgreich bearbeitet wurde, sollten Sie überprüfen, ob Sie Ihren Standardtexteditor mithilfe der Umgebungsvariable KUBE_EDITOR richtig konfiguriert haben. Weitere Informationen hierzu finden Sie unter Konfigurieren eines Texteditors für Kubectl.

9 Überprüfen Sie, ob der Cluster aktualisiert wurde.

v1alpha3-Cluster:

```
kubectl get tanzukubernetescluster
```

v1beta1-Cluster:

```
kubectl get cluster
```

Aktualisieren eines TKG-Clusters mithilfe der Tanzu-CLI

Aktualisieren Sie einen TKG-Cluster, indem Sie ein Upgrade der Tanzu Kubernetes-Version-Version mithilfe der Tanzu-CLI durchführen.

Sie können ein paralleles Update eines TKGS-Clusters initiieren, indem Sie mithilfe der Tanzu-CLI ein Upgrade der Tanzu Kubernetes-Version-Version durchführen.

Umfassende Informationen zur Nutzung finden Sie im *Tanzu-CLI-Referenzhandbuch*.

Voraussetzungen

Installieren der Tanzu-CLI zur Verwendung mit TKG-Dienst-Clustern.

Verfahren

1 Authentifizieren Sie sich beim Supervisor.

2 Listen Sie den TKG-Cluster auf.

```
tanzu cluster list
```

3 Aktualisieren Sie den TKG-Cluster.

```
tanzu cluster upgrade CLUSTER-NAME --tkr TKR-NAME -n VSPHERE-NAMESPACE
```

Dabei gilt:

- `CLUSTER-NAME` ist der Name des TKG-Clusters, den Sie als Ziel für das Upgrade verwenden.
- `TKR-NAME` ist die Zeichenfolge der TKR-Version.
- `VSPHERE-NAMESPACE` ist der Name des vSphere-Namespace, in dem der TKG-Cluster bereitgestellt wird.

Beispiel:

```
tanzu cluster upgrade tkg-cluster-1 --tkr v1.23.8---vmware.2-tkg.2-zshippable -n tkg2-cluster-ns
```

4 Bestätigen Sie das Cluster-Upgrade.

Nachdem das Upgrade des Clusters durchgeführt wurde, sollte Ihnen eine Meldung ähnlich der Folgenden angezeigt werden:

```
Cluster 'tkg-cluster-1' successfully upgraded to kubernetes version 'v1.23.8+vmware.2-tkg.2-zshippable'
```

Automatische Skalierung von TKG-Dienstclustern

10

Dieser Abschnitt enthält Informationen zur automatischen Skalierung von TKG-Dienstclustern.

Lesen Sie als Nächstes die folgenden Themen:

- Informationen zur automatischen Clusterskalierung
- Installieren der automatischen Skalierung des Clusters mithilfe von „kubectl“
- Installieren der automatischen Skalierung des Clusters über die Tanzu-CLI
- Upgrade eines automatisch skalierten Clusters mithilfe von Kubectl
- Upgrade eines automatisch skalierten Clusters mithilfe der Tanzu-CLI
- Testen der automatischen Skalierung des Clusters
- Löschen der automatischen Clusterskalierung

Informationen zur automatischen Clusterskalierung

Sie können die automatische Clusterskalierung bereitstellen, um die Anzahl der Worker-Knoten im TKG-Dienstcluster basierend auf den Anforderungen Ihrer Arbeitslasten automatisch anzupassen.

Informationen zur automatischen Clusterskalierung

Bei der automatischen Skalierung des TKG-Dienstclusters handelt es sich um eine Implementierung der automatischen Skalierung des Kubernetes-Clusters. Weitere Informationen finden Sie in der Dokumentation zur automatischen Clusterskalierung.

Die automatische Clusterskalierung unterstützt die horizontale und vertikale Skalierung von Clusterknoten. Bei Ausführung des Clusters auf einem Supervisor mit mehreren Zonen können einem bestimmten Verfügbarkeitsbereich zugewiesene Knotenpools automatisch skaliert werden.

Die automatische Clusterskalierung wird als Standardpaket bereitgestellt, das Sie mithilfe der Kubectl- oder Tanzu-CLI im Cluster installieren. Die automatische Clusterskalierung wird als Bereitstellung im TKG-Cluster mit den Anmeldedaten des Dienstkontos ausgeführt.

Es besteht eine 1:1-Beziehung zwischen der Nebenversion des Pakets für die automatische Skalierung und der TKr-Nebenversion. Wenn Sie beispielsweise TKr 1.27.11 verwenden, sollten Sie v1.27.2 der automatischen Skalierung installieren. Wenn die Version nicht übereinstimmt, schlägt der Paketabgleich fehl.

Während die automatische Clusterskalierung sowohl die horizontale als auch die horizontale Skalierung von Worker-Knoten unterstützt, gibt es einige Fälle, in denen Clusterknoten nicht herunterskaliert werden, da einige Arten von Anwendungen verhindern, dass Knoten herunterskaliert werden. Weitere Informationen finden Sie unter „Welche Pod-Typen können CA daran hindern, einen Knoten zu entfernen?“ in der Dokumentation der automatischen Clusterskalierung.

Versionsanforderungen

Für die automatische Clusterskalierung gelten die folgenden Versionsanforderungen.

- Die vSphere-Mindestversion ist vSphere 8 U3.
- Die TKr-Mindestversion ist TKr 1.27.x für vSphere 8.
- Die Nebenversion der TKr und die Nebenversion des Pakets für die automatische Clusterskalierung müssen mit übereinstimmen.

Paketanforderungen

Die automatische Clusterskalierung wird als Standardpaket bereitgestellt. Die Nebenversion des Pakets muss mit der Nebenversion der verwendeten TKr übereinstimmen. Wenn Sie beispielsweise TKr 1.27.11 verwenden, sollten Sie v1.27.2 der automatischen Skalierung installieren. Wenn die Version nicht übereinstimmt, schlägt der Paketabgleich fehl.

Möglicherweise müssen Sie das Zielpaket in einer nachfolgenden Repository-Version suchen. Beispielsweise befindet sich v1.27.2 der automatischen Skalierung in Version v2024.4.12 des Standardpaket-Repositorys. Spätere Versionen des Pakets für die automatische Skalierung, wie 1.28.x, 1.29.x, 1.30.x usw., befinden sich in nachfolgenden Repository-Versionen. Alle Standardpaket-Repositorys finden Sie, indem Sie den folgenden Befehl ausführen:

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

Workflow

Der allgemeine Workflow zum Aktivieren der automatischen Clusterskalierung lautet wie folgt:

1. Erstellen Sie einen neuen TKG-Cluster oder aktualisieren Sie einen vorhandenen TKG-Cluster mit Anmerkungen für die automatische Skalierung und entfernen Sie das Replikatfeld in `spec.topology.workers.machinedeployments`.
2. Installieren Sie das Paket-Repository im TKG-Cluster, den Sie erstellt oder aktualisiert haben.
3. Installieren Sie das Paket für die automatische Skalierung im TKG-Cluster, den Sie erstellt oder aktualisiert haben.

Die automatische Skalierung wird im TKG-Cluster als Bereitstellung im Namespace kube-system installiert.

Ausführliche Anleitungen finden Sie in folgenden Themen:

- Installieren der automatischen Skalierung des Clusters mithilfe von „kubectl“
- Installieren der automatischen Skalierung des Clusters über die Tanzu-CLI

Installieren der automatischen Skalierung des Clusters mithilfe von „kubectl“

Lesen Sie diese Anweisungen, um das Paket für die automatische Clusterskalierung mithilfe von „kubectl“ zu installieren und zu konfigurieren.

Anforderungen

Beachten Sie die folgenden Anforderungen.

- Die vSphere-Mindestversion ist vSphere 8 U3.
- Die TKr-Mindestversion ist TKr 1.27.x für vSphere 8.
- Die Nebenversion der TKr und die Nebenversion des Pakets für die automatische Clusterskalierung müssen mit übereinstimmen.

Achtung Es besteht eine 1:1-Beziehung zwischen der Nebenversion des Pakets für die automatische Skalierung und der TKr-Nebenversion. Wenn Sie beispielsweise TKr 1.27.11 verwenden, sollten Sie v1.27.2 der automatischen Skalierung installieren. Wenn die Version nicht übereinstimmt, schlägt der Paketabgleich fehl.

Konfigurieren des vSphere-Namespace

Schließen Sie die folgenden erforderlichen Aufgaben aus, um einen TKG-Cluster bereitzustellen.

1 Installieren oder aktualisieren Sie Ihre Umgebung auf vSphere 8 U3 und TKr 1.27.x für vSphere 8.

2 Erstellen oder aktualisieren Sie eine Inhaltsbibliothek mit den neuesten Tanzu Kubernetes-Versionen. Weitere Informationen hierzu finden Sie unter Kapitel 5 Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster.

3 Erstellen und konfigurieren Sie einen vSphere-Namespace zum Hosten des TKG-Clusters. Weitere Informationen hierzu finden Sie unter Kapitel 6 Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern.

4 Installieren Sie den Kubernetes-CLI-Tools für vSphere.

Das folgende Beispiel kann verwendet werden, um die Tools über die Befehlszeile zu installieren. Weitere Anleitungen finden Sie unter Installieren des Kubernetes-CLI-Tools für vSphere.

```
curl -LOk https://${SUPERVISOR_IP-or-FQDN}/wcp/plugin/linux-amd64/vsphere-plugin.zip
unzip vsphere-plugin.zip
mv -v bin/* /usr/local/bin/
```

5 Führen Sie `kubectl` und `kubectl vsphere` aus, um die Installation zu überprüfen.

Erstellen eines TKG-Clusters mit Anmerkungen für die automatische Skalierung

Befolgen Sie die Anweisungen, um einen TKG-Cluster zu erstellen. Weitere Anleitungen finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl.

Zum Verwenden der automatischen Skalierung müssen Sie den Cluster mit Bezeichnungsanmerkungen für die automatische Skalierung konfigurieren, wie im hier bereitgestellten Beispiel für die Clusterspezifikation dargestellt. Im Gegensatz zur regulären Clusterbereitstellung wird die Anzahl der Worker-Knotenreplikate nicht hart codiert. Kubernetes verfügt über eine integrierte Standardlogik für die Replikate, die auf den Anmerkungen zur Mindest- und Maximalgröße der automatischen Skalierung basiert. Da es sich um einen neuen Cluster handelt, wird die Mindestgröße zum Erstellen des Clusters verwendet. Weitere Informationen finden Sie unter https://cluster-api.sigs.k8s.io/tasks/automated-machine-management/autoscaling.

1 Authentifizieren Sie sich mithilfe von kubectl bei Supervisor.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-username USERNAME
```

2 Wechseln Sie den Kontext zum Ziel-vSphere-Namespace, der den Cluster hostet.

```
kubectl config use-context tkgs-cluster-namespace
```

3 Listen Sie die VM-Klassen auf, die im vSphere-Namespace verfügbar sind.

Sie können nur VM-Klassen verwenden, die an den Ziel-vSphere-Namespace gebunden sind. Weitere Informationen hierzu finden Sie unter Verwenden von VM-Klassen mit TKG-Dienstclustern.

4 Listen Sie die verfügbaren Klassen persistenter Speicher-Volumes auf.

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

Der Befehl gibt Details zum vSphere-Namespace zurück, einschließlich der Speicherklasse. Der Befehl `kubectl describe storageclasses` gibt auch verfügbare Speicherklassen zurück, erfordert jedoch vSphere-Administratorberechtigungen.

5 Listen Sie die verfügbaren Tanzu Kubernetes-Versionen auf.

```
kubectl get tkr
```

Dieser Befehl gibt die in diesem vSphere-Namespace verfügbaren TKrs und deren Kompatibilität zurück. Weitere Informationen hierzu finden Sie unter Kapitel 5 Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster.

6 Verwenden Sie die Informationen, die Sie in Erfahrung haben, um eine YAML-Datei für die TKG-Clusterspezifikation mit der erforderlichen Konfiguration für die automatische Clusterskalierung zu erstellen.

- Verwenden Sie die Anmerkungen `*-min-size` und `*-max-size` für die nodePools des Workers. In diesem Beispiel können zwischen 3 und 5 Worker-Knoten skaliert werden. Standardmäßig wird der Cluster mit 3 Worker-Knoten erstellt.
- Verwenden Sie die passende Nebenversion für die TKr und für das Paket für die automatische Skalierung.
- Die Werte `metadata.name` und `metadata.namespace` für den Cluster stimmen mit den Standardwerten des Pakets für die automatische Skalierung überein. Wenn Sie diese Werte in der Clusterspezifikation ändern, müssen Sie sie in `autoscaler-data-values` ändern (siehe unten).

```
#cc-autoscaler.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
 name: gc1
 namespace: cluster
spec:
 clusterNetwork:
   pods:
     cidrBlocks:
     - 192.0.2.0/16
   serviceDomain: cluster.local
   services:
     cidrBlocks:
     - 198.51.100.0/12
 topology:
   class: tanzukubernetescluster
   controlPlane:
     metadata: {}
     replicas: 3
   variables:
   - name: storageClasses
     value:
     - wcpglobal-storage-profile
   - name: vmClass
     value: guaranteed-medium
   - name: storageClass
     value: wcpglobal-storage-profile
   #minor versions must match
```

```
    version: v1.27.11---vmware.1-fips.1-tkg.2
    workers:
      machineDeployments:
      - class: node-pool
        metadata:
          annotations:
            cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size: "3"
            cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size: "5"
        name: np-1
```

7 Übernehmen Sie die Clusterspezifikation.

```
kubectl apply -f cc-autoscaler.yaml
```

8 Überprüfen Sie die Clustererstellung.

```
kubectl get cluster,vm
```

9 Überprüfen Sie die Clusterknotenversion.

```
kubectl get node
```

Installieren des Paketmanagers auf dem TKG-Cluster

Sobald der TKG-Cluster bereitgestellt ist, installieren Sie den Paketmanager auf dem Cluster, und richten Sie das Paket-Repository ein.

1 Melden Sie sich beim von Ihnen bereitgestellten TKG-Cluster an.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN \
--vsphere-username USERNAME \
--tanzu-kubernetes-cluster-name CLUSTER-NAME \
--tanzu-kubernetes-cluster-namespace NAMESPACE-NAME
```

2 Installieren Sie das Carvel-Tool imgpkg.

```
wget -O- https://carvel.dev/install.sh > install.sh
sudo bash install.sh
```

3 Führen Sie `imgpkg version` aus, um die Installation zu überprüfen.

4 Überprüfen Sie die Paket-Repository-Version.

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

5 Installieren Sie das Paketrepository. Aktualisieren Sie die Repository-Version entsprechend.

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageRepository
metadata:
  name: tanzu-standard
  namespace: tkg-system
```

```
spec:
  fetch:
    imgpkgBundle:
      image: projects.registry.vmware.com/tkg/packages/standard/repo:v2024.4.12
```

6 Überprüfen Sie das Paketrepository.

```
kubectl get packagerepository -A
NAMESPACE    NAME             AGE     DESCRIPTION
tkg-system   tanzu-standard   2m22s   Reconcile succeeded
```

7 Überprüfen Sie, ob das Paket für die automatische Clusterskalierung vorhanden ist.

```
kubectl get package
NAME                                                                    PACKAGEMETADATA
NAME                                VERSION                 AGE
cert-manager.tanzu.vmware.com.1.7.2+vmware.3-tkg.1                      cert-
manager.tanzu.vmware.com                    1.7.2+vmware.3-tkg.1    5s
cert-manager.tanzu.vmware.com.1.7.2+vmware.3-tkg.3                      cert-
manager.tanzu.vmware.com                    1.7.2+vmware.3-tkg.3    5s
cluster-autoscaler.tanzu.vmware.com.1.25.1+vmware.1-tkg.3               cluster-
autoscaler.tanzu.vmware.com              1.25.1+vmware.1-tkg.3   5s
cluster-autoscaler.tanzu.vmware.com.1.26.2+vmware.1-tkg.3               cluster-
autoscaler.tanzu.vmware.com              1.26.2+vmware.1-tkg.3   5s
cluster-autoscaler.tanzu.vmware.com.1.27.2+vmware.1-tkg.3               cluster-
autoscaler.tanzu.vmware.com              1.27.2+vmware.1-tkg.3   5s
contour.tanzu.vmware.com.1.26.2+vmware.1-tkg.1
contour.tanzu.vmware.com                          1.26.2+vmware.1-tkg.1   5s
...
```

Installieren des Pakets für die automatische Skalierung

Jetzt können Sie das Paket für die automatische Clusterskalierung installieren. Die automatische Clusterskalierung wird als Bereitstellung im Namespace `kube-system` installiert.

1 Erstellen Sie die Konfigurationsdatei `autoscaler.yaml`.

- Sie können die automatische Skalierung anpassen. Ändern Sie dazu den Abschnitt `autoscaler-data-values` der Spezifikation mit den für Ihre Umgebung geeigneten Werten.

```
#autoscaler.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: autoscaler-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: autoscaler-role-binding
roleRef:
```

```
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: autoscaler-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: autoscaler
  namespace: tkg-system
spec:
  serviceAccountName: autoscaler-sa
  packageRef:
    refName: cluster-autoscaler.tanzu.vmware.com
    versionSelection:
      constraints: 1.27.2+vmware.1-tkg.3
  values:
  - secretRef:
      name: autoscaler-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: autoscaler-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    arguments:
      ignoreDaemonsetsUtilization: true
      maxNodeProvisionTime: 15m
      maxNodesTotal: 0
      metricsPort: 8085
      scaleDownDelayAfterAdd: 10m
      scaleDownDelayAfterDelete: 10s
      scaleDownDelayAfterFailure: 3m
      scaleDownUnneededTime: 10m
    clusterConfig:
      clusterName: "gc1"
      clusterNamespace: "cluster"
    paused: false
```

2 Installieren Sie das Paket für die automatische Clusterskalierung.

```
kubectl apply -f autoscaler.yaml
```

3 Überprüfen Sie die Installation des Pakets für die automatische Skalierung.

```
kubectl get pkgi -A | grep autoscaler
```

Erwartetes Ergebnis:

```
tkg-system autoscaler cluster-autoscaler.tanzu.vmware.com 1.27.2+vmware.1-tkg.3 Reconcile
succeeded 3m52s
```

4 Überprüfen Sie die Bereitstellung für die automatische Skalierung.

```
kubectl get pods -n kube-system | grep autoscaler
```

```
cluster-autoscaler-798b65bd9f-bht8n 1/1 Running 0 2m
```

Testen der automatischen Skalierung des Clusters

Stellen Sie zum Testen der automatischen Skalierung des Clusters eine Anwendung bereit, erhöhen Sie die Anzahl der Replikate, und überprüfen Sie, ob zusätzliche Worker-Knoten horizontal skaliert werden, um die Last zu bewältigen.

Weitere Informationen hierzu finden Sie unter Testen der automatischen Skalierung des Clusters.

Upgrade eines automatisch skalierten Clusters

Damit Sie ein Upgrade eines automatisch skalierten Clusters durchführen können, halten Sie das Paket für die automatische Skalierung an.

Weitere Informationen hierzu finden Sie unter Upgrade eines automatisch skalierten Clusters mithilfe von Kubectl.

Installieren der automatischen Skalierung des Clusters über die Tanzu-CLI

Lesen Sie diese Anweisungen, um das Paket für die automatische Clusterskalierung über die Tanzu-CLI zu installieren und zu konfigurieren.

Anforderungen

Beachten Sie die folgenden Anforderungen.

- Die vSphere-Mindestversion ist vSphere 8 U3, einschließlich vCenter und ESXi-Hosts.
- Die TKr-Mindestversion ist TKr 1.27.x für vSphere 8.
- Die Nebenversion der TKr und die Nebenversion des Pakets für die automatische Clusterskalierung müssen mit übereinstimmen.

Hinweis Es besteht eine 1:1-Beziehung zwischen der Nebenversion des Pakets für die automatische Skalierung und der TKr-Nebenversion. Wenn Sie beispielsweise TKr 1.27.11 verwenden, sollten Sie v1.27.2 der automatischen Skalierung installieren. Wenn die Version nicht übereinstimmt, schlägt der Paketabgleich fehl.

Konfigurieren des vSphere-Namespace

Schließen Sie die folgenden erforderlichen Aufgaben aus, um einen TKG-Cluster bereitzustellen.

1 Installieren oder aktualisieren Sie Ihre Umgebung auf vSphere 8 U3 und TKr 1.27.x für vSphere 8.

2 Erstellen oder aktualisieren Sie eine Inhaltsbibliothek mit den neuesten Tanzu Kubernetes-Versionen. Weitere Informationen hierzu finden Sie unter Kapitel 5 Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster.

3 Erstellen und konfigurieren Sie einen vSphere-Namespace zum Hosten des TKG-Clusters. Weitere Informationen hierzu finden Sie unter Kapitel 6 Konfigurieren von vSphere-Namespaces für das Hosting von TKG-Dienst-Clustern.

4 Installieren Sie den Kubernetes-CLI-Tools für vSphere.

   Das folgende Beispiel kann verwendet werden, um die Tools über die Befehlszeile zu installieren. Weitere Anleitungen finden Sie unter Installieren des Kubernetes-CLI-Tools für vSphere.

```
wget https://SUPERVISOR-IP-or-FQDN/wcp/plugin/linux-amd64/vsphere-plugin.zip
unzip vsphere-plugin.zip
chmod +x bin/kubectl*
mv bin/kubectl* /usr/bin/kubectl vsphere --help
rm ~/.kube/config
kubectl vsphere login --insecure-skip-tls-verify --server SUPERVISOR-IP-or-FQDN --tanzu-
kubernetes-cluster-namespace VSPHERE-NAMESPACE --vsphere-username VSPHERE-USER
kubectl config use-context VSPHERE-NAMESPACE
```

5 Führen Sie `kubectl` und `kubectl vsphere` aus, um die Installation zu überprüfen.

Erstellen eines TKG-Clusters mit Anmerkungen für die automatische Skalierung

Befolgen Sie die Anweisungen, um einen TKG-Cluster zu erstellen. Weitere Anleitungen finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl.

Zum Verwenden der automatischen Skalierung müssen Sie den Cluster mit Bezeichnungsanmerkungen für die automatische Skalierung konfigurieren, wie im hier bereitgestellten Beispiel für die Clusterspezifikation dargestellt. Im Gegensatz zur regulären Clusterbereitstellung wird die Anzahl der Worker-Knotenreplikate nicht hart codiert. Kubernetes verfügt über eine integrierte Standardlogik für die Replikate, die auf den Anmerkungen

zur Mindest- und Maximalgröße der automatischen Skalierung basiert. Da es sich um einen neuen Cluster handelt, wird die Mindestgröße zum Erstellen des Clusters verwendet. Weitere Informationen finden Sie unter https://cluster-api.sigs.k8s.io/tasks/automated-machine-management/autoscaling.

1 Authentifizieren Sie sich mithilfe von kubectl bei Supervisor.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-username USERNAME
```

2 Wechseln Sie den Kontext zum Ziel-vSphere-Namespace, der den Cluster hostet.

```
kubectl config use-context tkgs-cluster-namespace
```

3 Listen Sie die VM-Klassen auf, die im vSphere-Namespace verfügbar sind.

Sie können nur VM-Klassen verwenden, die an den Ziel-vSphere-Namespace gebunden sind. Weitere Informationen hierzu finden Sie unter Verwenden von VM-Klassen mit TKG-Dienstclustern.

4 Listen Sie die verfügbaren Klassen persistenter Speicher-Volumes auf.

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

Der Befehl gibt Details zum vSphere-Namespace zurück, einschließlich der Speicherklasse. Der Befehl `kubectl describe storageclasses` gibt auch verfügbare Speicherklassen zurück, erfordert jedoch vSphere-Administratorberechtigungen.

5 Listen Sie die verfügbaren Tanzu Kubernetes-Versionen auf.

```
kubectl get tkr
```

Dieser Befehl gibt die in diesem vSphere-Namespace verfügbaren TKrs und deren Kompatibilität zurück. Weitere Informationen hierzu finden Sie unter Kapitel 5 Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster.

6 Verwenden Sie die Informationen, die Sie in Erfahrung haben, um eine YAML-Datei für die TKG-Clusterspezifikation mit der erforderlichen Konfiguration für die automatische Clusterskalierung zu erstellen.

- Verwenden Sie die Anmerkungen `*-min-size` und `*-max-size` für die nodePools des Workers. In diesem Beispiel können zwischen 3 und 5 Worker-Knoten skaliert werden. Standardmäßig wird der Cluster mit 3 Worker-Knoten erstellt.
- Verwenden Sie die passende Nebenversion für die TKr und für das Paket für die automatische Skalierung.

- Die Werte `metadata.name` und `metadata.namespace` für den Cluster stimmen mit den Standardwerten des Pakets für die automatische Skalierung überein. Wenn Sie diese Werte in der Clusterspezifikation ändern, müssen Sie sie in `autoscaler-data-values` ändern (siehe unten).

```
#cc-autoscaler.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
 name: tkc
 namespace: cluster
spec:
 clusterNetwork:
   pods:
     cidrBlocks:
     - 192.0.2.0/16
   serviceDomain: cluster.local
   services:
     cidrBlocks:
     - 198.51.100.0/12
 topology:
   class: tanzukubernetescluster
   controlPlane:
     metadata: {}
     replicas: 3
   variables:
   - name: storageClasses
     value:
     - wcpglobal-storage-profile
   - name: vmClass
     value: guaranteed-medium
   - name: storageClass
     value: wcpglobal-storage-profile
   #minor versions must match
   version: v1.27.11---vmware.1-fips.1-tkg.2
   workers:
     machineDeployments:
     - class: node-pool
       metadata:
         annotations:
           cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size: "3"
           cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size: "5"
       name: np-1
```

7 Übernehmen Sie die Clusterspezifikation.

```
kubectl apply -f cc-autoscaler.yaml
```

8 Überprüfen Sie die Clustererstellung.

```
kubectl get cluster,vm
```

9 Überprüfen Sie die Clusterknotenversion.

```
kubectl get node
```

Erstellen des Paket-Repositorys auf dem TKG-Cluster

Sobald der TKG-Cluster bereitgestellt wurde, installieren Sie die Tanzu-CLI, und richten Sie das Paket-Repository ein.

1 Installieren Sie die Tanzu-CLI.

Weitere Informationen hierzu finden Sie unter Installieren der Tanzu-CLI zur Verwendung mit TKG-Dienst-Clustern.

2 Melden Sie sich beim Cluster an.

```
rm ~/.kube/config
kubectl vsphere login --insecure-skip-tls-verify --server 192.168.0.2 --tanzu-kubernetes-
cluster-namespace autoscaler --vsphere-username administrator@vsphere.local --tanzu-
kubernetes-cluster-name cckubectl
config use-context cc
```

3 Erstellen Sie das Paket-Repository.

```
#Standard package repository URL might change depending on the required cluster autoscaler
version
tanzu package repository add standard-repo --url projects.registry.vmware.com/tkg/packages/
standard/repo:v2024.4.12 -n tkg-system
tanzu package available list -n tkg-system
tanzu package available get cluster-autoscaler.tanzu.vmware.com -n tkg-system
```

Installieren des Pakets für die automatische Skalierung

Installieren Sie das Paket für die automatische Clusterskalierung. Die automatische Clusterskalierung wird im Namespace „`kube-system`“ installiert.

1 Generieren Sie die `values.yaml`-Standarddatei mithilfe des Tanzu-CLI-Befehls.

```
tanzu package available get cluster-autoscaler.tanzu.vmware.com/1.27.2+vmware.1-tkg.3  -n
tkg-system --default-values-file-output values.yaml
```

2 Aktualisieren Sie die `values.yaml` für die Paketinstallation.

```
arguments:
  ignoreDaemonsetsUtilization: true
  maxNodeProvisionTime: 15m
  maxNodesTotal: 0
  metricsPort: 8085
  scaleDownDelayAfterAdd: 10m
  scaleDownDelayAfterDelete: 10s
  scaleDownDelayAfterFailure: 3m
  scaleDownUnneededTime: 10m
```

```
clusterConfig:
  clusterName: "tkc"
  clusterNamespace: "cluster"
paused: false
```

3 Installieren Sie das Paket für die automatische Clusterskalierung über die Tanzu-CLI.

```
tanzu package install cluster-autoscaler-pkgi -n tkg-system --package cluster-
autoscaler.tanzu.vmware.com --version 1.27.2+vmware.1-tkg.3 --values-file values.yaml
```

Testen der automatischen Skalierung des Clusters

Stellen Sie zum Testen der automatischen Skalierung des Clusters eine Anwendung bereit, erhöhen Sie die Anzahl der Replikate, und überprüfen Sie, ob zusätzliche Worker-Knoten horizontal skaliert werden, um die zusätzliche Last zu bewältigen.

Weitere Informationen hierzu finden Sie unter Testen der automatischen Skalierung des Clusters.

Upgrade eines automatisch skalierten Clusters

Damit Sie ein Upgrade eines automatisch skalierten Clusters durchführen können, müssen Sie zuerst das Paket für die automatische Skalierung anhalten.

Weitere Informationen hierzu finden Sie unter Upgrade eines automatisch skalierten Clusters mithilfe der Tanzu-CLI.

Upgrade eines automatisch skalierten Clusters mithilfe von Kubectl

Vor dem Upgrade eines TKG-Clusters muss die automatische Skalierung angehalten werden. Nach dem Upgrade der TKr-Version des Clusters müssen Sie die Paketversion für die automatische Skalierung so aktualisieren, dass sie mit der TKr-Nebenversion übereinstimmt.

Anforderungen

Bei dieser Aufgabe wird davon ausgegangen, dass die automatische Clusterskalierung auf einem TKG-Cluster installiert wurde. Weitere Informationen hierzu finden Sie unter Installieren der automatischen Skalierung des Clusters mithilfe von „kubectl“.

Vor dem Cluster-Upgrade: Anhalten der automatischen Skalierung

Bevor Sie mit der installierten automatischen Skalierung ein Upgrade eines TKG-Clusters durchführen, müssen Sie zuerst das Paket für die automatische Skalierung anhalten.

1 Halten Sie das Paket für die automatische Clusterskalierung an. Legen Sie dazu im geheimen Schlüssel `autoscaler-data-values.yaml` den booleschen Wert `paused` auf `true` fest.

```
---
apiVersion: v1
kind: Secret
metadata:
  name: autoscaler-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    arguments:
      ignoreDaemonsetsUtilization: true
      maxNodeProvisionTime: 15m
      maxNodesTotal: 0
      metricsPort: 8085
      scaleDownDelayAfterAdd: 10m
      scaleDownDelayAfterDelete: 10s
      scaleDownDelayAfterFailure: 3m
      scaleDownUnneededTime: 10m
    clusterConfig:
      clusterName: "gc1"
      clusterNamespace: "cluster"
    paused: true
```

2 Wenden Sie die Aktualisierungen auf den geheimen Schlüssel `autoscaler-data-values` an.

```
kubectl apply -f autoscaler-data-values.yaml
```

Upgrade des Clusters

Sobald die automatische Skalierung angehalten wurde, fahren Sie mit der Aktualisierung des Clusters fort.

1 Aktualisieren Sie die Kubernetes-Version des TKG-Clusters.

Weitere Informationen hierzu finden Sie unter Aktualisieren eines TKG-Clusters durch Bearbeiten der TKR-Version.

Nach dem Cluster-Upgrade: Aktualisieren der Paketversion für die automatische Skalierung

Aktualisieren Sie nach dem Upgrade des Clusters die Paketversion für die automatische Skalierung entsprechend der TKr-Nebenversion, und deaktivieren Sie die Pause.

1 Wählen Sie die entsprechende Version der automatischen Skalierung aus.

Die Nebenversionen der TKr und des Pakets für die automatische Skalierung müssen übereinstimmen. Wenn Sie beispielsweise ein Upgrade des Clusters auf TKr v1.28.8 durchgeführt haben, müssen Sie das v1.28.x-Paket für die automatische Skalierung verwenden.

2 Aktualisieren Sie Ressourcen für die automatische Skalierung. Legen Sie dazu die Zielversion der automatischen Skalierung fest, und setzen Sie den angehaltenen Schlüssel auf „false“ zurück.

```
#autoscaler-package-upgrade.yaml
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: autoscaler
  namespace: tkg-system
spec:
  serviceAccountName: autoscaler-sa
  packageRef:
    refName: cluster-autoscaler.tanzu.vmware.com
    versionSelection:
      constraints: 1.28.0+vmware.1-tkg.1
  values:
  - secretRef:
      name: autoscaler-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: autoscaler-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    arguments:
      ignoreDaemonsetsUtilization: true
      maxNodeProvisionTime: 15m
      maxNodesTotal: 0
      metricsPort: 8085
      scaleDownDelayAfterAdd: 10m
      scaleDownDelayAfterDelete: 10s
      scaleDownDelayAfterFailure: 3m
      scaleDownUnneededTime: 10m
    clusterConfig:
      clusterName: "gc1"
      clusterNamespace: "cluster"
    paused: false
```

3 Wenden Sie die Aktualisierungen auf das Paket für die automatische Skalierung an.

```
kubectl apply -f autoscaler-package-upgrade.yaml
```

4 Überprüfen Sie, ob der Pod für die automatische Skalierung im kube-system-Namespace ausgeführt wird.

5 Testen Sie die automatische Skalierung des Clusters.

Testen der automatischen Skalierung des Clusters.

Upgrade eines automatisch skalierten Clusters mithilfe der Tanzu-CLI

Vor dem Upgrade eines TKG-Clusters muss die automatische Skalierung angehalten werden. Nach dem Upgrade der TKr-Version des Clusters müssen Sie die Paketversion für die automatische Skalierung so aktualisieren, dass sie mit der TKr-Nebenversion übereinstimmt.

Anforderungen

Bei dieser Aufgabe wird davon ausgegangen, dass die automatische Clusterskalierung auf einem TKG-Cluster installiert wurde. Weitere Informationen hierzu finden Sie unter Installieren der automatischen Skalierung des Clusters über die Tanzu-CLI.

Vor dem Cluster-Upgrade: Anhalten der automatischen Skalierung

Bevor Sie mit der installierten automatischen Skalierung ein Upgrade eines TKG-Clusters durchführen, müssen Sie zuerst das Paket für die automatische Skalierung anhalten.

1 Halten Sie das Paket für die automatische Clusterskalierung an. Legen Sie dazu in der Konfigurationsdatei `values.yaml` den booleschen Wert `paused` auf `true` fest.

```
arguments:
  ignoreDaemonsetsUtilization: true
  maxNodeProvisionTime: 15m
  maxNodesTotal: 0
  metricsPort: 8085
  scaleDownDelayAfterAdd: 10m
  scaleDownDelayAfterDelete: 10s
  scaleDownDelayAfterFailure: 3m
  scaleDownUnneededTime: 10m
clusterConfig:
  clusterName: "tkc"
  clusterNamespace: "cluster"
paused: true #set to true before upgrade
```

2 Aktualisieren Sie das Paket mithilfe der Tanzu CLI.

```
tanzu package installed update cluster-autoscaler-pkgi -n tkg-system --package cluster-
autoscaler.tanzu.vmware.com --values-file values.yaml
```

Upgrade des Clusters

Sobald die automatische Skalierung angehalten wurde, fahren Sie mit der Aktualisierung des Clusters fort.

1 Aktualisieren Sie die Kubernetes-Version des TKG-Clusters.

   Weitere Informationen hierzu finden Sie unter Aktualisieren eines TKG-Clusters durch Bearbeiten der TKR-Version.

Nach dem Cluster-Upgrade: Aktualisieren der Paketversion für die automatische Skalierung

Aktualisieren Sie nach dem Upgrade des Clusters die Version des Pakets für die automatische Skalierung, damit sie mit der TKr-Nebenversion übereinstimmt, und setzen Sie den angehaltenen Schlüssel auf „false“ zurück.

1 Wählen Sie die entsprechende Version der automatischen Skalierung aus.

   Die Nebenversionen der TKr und des Pakets für die automatische Skalierung müssen übereinstimmen. Wenn Sie beispielsweise ein Upgrade des Clusters auf TKr v1.28.8 durchgeführt haben, müssen Sie das v1.28.0-Paket für die automatische Skalierung verwenden.

2 Generieren Sie die `values.yaml`-Standarddatei mithilfe des Tanzu-CLI-Befehls.

```
tanzu package available get cluster-autoscaler.tanzu.vmware.com/1.28.0+vmware.1-tkg.1  -n
tkg-system --default-values-file-output new-values.yaml
```

3 Aktualisieren Sie die Datei `new-values.yaml` mit der neuen Paketversion, und setzen Sie den angehaltenen Schlüssel auf „false“ zurück.

4 Verwenden Sie die Tanzu-CLI, um die Installation der automatischen Skalierung des Clusters zu aktualisieren.

```
tanzu package installed update cluster-autoscaler-pkgi -n tkg-system --package cluster-
autoscaler.tanzu.vmware.com --values-file new-values.yaml --version 1.28.1+vmware.1-tkg.1
```

Testen der automatischen Skalierung des Clusters

Lesen Sie diese Anweisungen, um die installierte automatische Skalierung des Clusters zu testen.

Anforderungen

Bei dieser Aufgabe wird davon ausgegangen, dass die automatische Clusterskalierung auf einem TKG-Cluster installiert wurde.

- Installieren der automatischen Skalierung des Clusters mithilfe von „kubectl“
- Installieren der automatischen Skalierung des Clusters über die Tanzu-CLI

Testen der automatischen Skalierung des Clusters

Wenn Sie überprüfen möchten, ob die automatische Skalierung die Worker-Knoten automatisch skaliert, stellen Sie eine Anwendung bereit, und skalieren Sie dann die Anzahl der Replikate in der Bereitstellung. Die automatische Skalierung führt eine vertikale Hochskalierung der Worker-Knoten durch, sofern die Knotenressourcen nicht ausreichen.

1 Erstellen Sie die folgende Anwendungsdefinition mit dem Namen `app.yaml`.

```
apiVersion: v1
kind: Namespace
metadata:
  name: app
  labels:
    pod-security.kubernetes.io/enforce: privileged
---
apiVersion: v1
kind: Service
metadata:
  name: application-cpu
  namespace: app
  labels:
    app: application-cpu
spec:
  type: ClusterIP
  selector:
    app: application-cpu
  ports:
    - protocol: TCP
      name: http
      port: 80
      targetPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: application-cpu
  namespace: app
  labels:    app: application-cpu
spec:
  selector:
    matchLabels:
      app: application-cpu
    replicas: 1
    strategy:
      type: RollingUpdate
      rollingUpdate:
        maxSurge: 1
        maxUnavailable: 0
    template:
      metadata:
        labels:
          app: application-cpu
      spec:
```

```
        containers:
        - name: application-cpu
          image: wcp-docker-ci.artifactory.eng.vmware.com/app-cpu:v1.0.0
          imagePullPolicy: Always
          ports:
            - containerPort: 80
          resources:
            requests:
              memory: 50Mi
              cpu: 500m
            limits:
              memory: 500Mi
              cpu: 2000m
```

2 Erstellen Sie die Anwendung.

```
kubectl apply -f app.yaml
```

3 Skalieren Sie die Replikate der Anwendung vertikal hoch, um die automatische Skalierung auszulösen.

Erhöhen Sie beispielsweise die Anzahl der `spec.selector.replicas` von „1“ auf eine größere Anzahl, sodass zusätzliche Worker-Knoten erforderlich sind.

4 Aktualisieren Sie die Anwendung.

```
kubectl apply -f app.yaml
```

5 Überprüfen Sie, ob zusätzliche Worker-Knoten erstellt wurden, um die Last zu bewältigen.

Die automatische Skalierung führt eine vertikale Hochskalierung der Worker-Knoten durch, sobald die Knotenressourcen nicht ausreichen.

Löschen der automatischen Clusterskalierung

In dieser Anleitung erfahren Sie, wie Sie eine installierte automatische Clusterskalierung löschen.

Anforderungen

Bei dieser Aufgabe wird davon ausgegangen, dass die automatische Clusterskalierung auf einem TKG-Cluster installiert wurde.

- Installieren der automatischen Skalierung des Clusters mithilfe von „kubectl“
- Installieren der automatischen Skalierung des Clusters über die Tanzu-CLI

Löschen der automatischen Clusterskalierung mit Kubectl

1 Führen Sie den folgenden Befehl aus, um die automatische Clusterskalierung mit Kubectl zu löschen.

```
kubectl delete -f autoscaler.yaml
```

Hinweis `autoscaler.yaml` ist der Name, den Sie bei der Bereitstellung des Pakets für die automatische Clusterskalierung verwendet haben. Passen Sie den Befehl entsprechend an, falls Sie einen anderen Namen verwendet haben. Weitere Informationen hierzu finden Sie unter Installieren der automatischen Skalierung des Clusters mithilfe von „kubectl“.

Löschen der automatischen Clusterskalierung mit der Tanzu CLI

1 Führen Sie den folgenden Befehl aus, um die automatische Clusterskalierung mit der Tanzu CLI zu löschen.

```
tanzu package installed delete -n tkg-system cluster-autoscaler-pkgi
```

Installieren von Standardpaketen auf TKG-Dienst-Clustern

11

VMware stellt einen Standardsatz von Open Source-Anwendungspaketen bereit, die in Ihren TKG-Dienst-Clustern zur Inbetriebnahme installiert werden können.

Lesen Sie als Nächstes die folgenden Themen:

- Standardpakete für TKG-Dienst-Cluster
- Installieren von Standardpaketen auf einem TKG-Cluster mithilfe der TKr für vSphere 8.x
- Standardpaketreferenz
- Installieren von Standardpaketen in einem TKG-Cluster mithilfe von TKr für vSphere 7.x

Standardpakete für TKG-Dienst-Cluster

Die vSphere IaaS control plane unterstützt Standardpakete für die Installation in TKG-Dienst-Clustern.

Unterstützte Pakete für TKG-Dienstcluster

In der Tabelle werden die Standardpakete aufgelistet, die für die Installation auf TKG-Dienst-Clustern verfügbar sind, die mit TKrs für vSphere 8.x bereitgestellt werden. Erfüllen Sie alle erforderlichen Voraussetzungen, bevor Sie das Zielpaket installieren.

Paket	Beschreibung	Anleitung
Cert Manager	Zertifikatsverwaltung	Installieren des Zertifikatmanagers
Contour mit Envoy	Kubernetes-Ingress-Controller und Reverse-Proxy	Installieren von Contour mit Envoy
ExternalDNS	DNS-Lookup von Kubernetes-Diensten	Installieren von ExternalDNS
Fluent Bit	Protokollweiterleitung	Installieren von ExternalDNS
Prometheus mit Alertmanager	Überwachung und Warnungen	Installieren von Prometheus mit Alertmanager
Grafana	Visualisierung	Installieren von Grafana
Harbor	Container-Registrierung	Installieren der Harbor-Registrierung

Paket	Beschreibung	Anleitung
Webhook für die Validierung des externen CSI-Snapshots vSphere PV-CSI-Webhook	Snapshots der persistenten Speicherung	Kapitel 15 Erstellen von Snapshots in einem TKG-Dienst-Cluster
Automatischen Skalierung des Clusters	Automatische Skalierung von Clustern	Kapitel 10 Automatische Skalierung von TKG-Dienstclustern

Installieren von Standardpaketen auf einem TKG-Cluster mithilfe der TKr für vSphere 8.x

In diesem Abschnitt finden Sie Informationen zum Installieren der Standardpakete in einem TKG-Dienst-Cluster, der mit einer TKr für vSphere 8.x bereitgestellt wird.

Allgemeine Anforderungen

Halten Sie diese allgemeinen Anforderungen für die Installation von Standardpaketen auf einem TKG-Dienst-Cluster ein.

Plattformanforderungen

Diese Anweisungen gelten besonders für die Installation von Standardpaketen auf TKG-Clustern, die mit TKrs für vSphere 8.x bereitgestellt werden. Weitere Informationen finden Sie in den Versionshinweisen zu TKr.

Wenn Sie Standardpakete auf TKG-Clustern bereitstellen, die mit TKr für vSphere 7.x bereitgestellt wurden, finden Sie weitere Informationen unter Installieren von Standardpaketen in einem TKG-Cluster mithilfe von TKr für vSphere 7.x.

Repository-Anforderungen

vSphere IaaS control plane unterstützt die Installation von Standardpaketen auf TKG-Clustern für vSphere 8-kompatible TKrs. Das auf vSphere 8 Verwenden von Kubernetes-Versionen mit TKG-Dienstclustern umfasst das Carvel-Paketsystem sowie den Kapp-Controller. Beide Komponenten werden automatisch als Teil des TKr-Images verwaltet, auf dem TKG-Knoten basieren. Informationen zur TKr-Kompatibilität mit vSphere finden Sie in den Versionshinweisen zu TKr.

Clientanforderungen

Für die Installation von Standardpaketen auf TKG-Clustern, die mit TKrs für vSphere 8.x bereitgestellt werden, sind Kubernetes-CLI-Tools für vSphere erforderlich. Hierzu gehören Kubectl, das vSphere-Plug-In für kubectl und die Tanzu-CLI. Informationen zur Installation dieser Tools finden Sie unter Installieren von CLI-Tools für TKG-Dienst-Cluster.

Speicheranforderungen

Der TKG-Cluster, in dem Sie Standardpakete bereitstellen, sollte mit einer Standardspeicherklasse bereitgestellt werden. Insbesondere erfordern die Prometheus- und Grafana-Pakete eine Standardspeicherklasse. Wenn Sie einen TKG-Cluster bereitgestellt haben, ohne die Standardspeicherklasse anzugeben, können Sie die vorhandene Speicherklasse patchen und die erforderliche Anmerkung hinzufügen, um sie als Standard anzugeben. Weitere Informationen finden Sie unter Patchen einer Speicherklasse.

Der Speichergrenzwert für den vSphere-Namespace, auf dem der TKG-Cluster bereitgestellt wird, auf dem Sie die Tanzu-Pakete installieren, muss größer als die Gesamtanforderung eines dauerhaften Datenträgers sein. Weitere Informationen zum vSphere-Namespace-Speicherkontingent finden Sie unter Konfigurieren eines vSphere-Namespace für TKG-Dienst-Cluster.

Tabelle 11-1. Anforderungen an persistenten Speicher für Standardpakete

Komponente	TKG-Erweiterung	Standardm. Speichergröße
Grafana	Grafana	8 Gi
Prometheus-Server	Prometheus	8 Gi
Alertmanager	Prometheus	8 Gi
Harbor	Harbor-Registrierung	Variiert nach PVC

So passen Sie den Speichergrenzwert für den vSphere-Namespace an, in dem der TKG-Cluster bereitgestellt wird:

1. Melden Sie sich mithilfe des vSphere Client beim vCenter Server an, auf dem vSphere IaaS control plane aktiviert ist.
2. Wählen Sie den vSphere-Namespace aus, auf dem der Tanzu Kubernetes-Zielcluster bereitgestellt wird.
3. Wählen Sie **Konfigurieren > Ressourcengrenzwerte** aus.
4. Klicken Sie auf **Bearbeiten**.
5. Passen Sie den **Speichergrenzwert** so an, dass er größer als die Gesamtbeanspruchungen dauerhafter Volumes ist, die für die Prometheus- und Grafana-Erweiterungen benötigt werden.

Erstellen des Paket-Repositorys

Folgen Sie den Anweisungen, um auf einem TKG-Dienst-Cluster, auf dem TKr für vSphere 8.x ausgeführt wird, das Standardpaket-Repository einzurichten.

Anforderungen

Beachten Sie die folgenden Anforderungen, bevor Sie das Paket-Repository erstellen.

- Allgemeine Anforderungen

- Installieren des Kubernetes-CLI-Tools für vSphere
- Installieren der Tanzu-CLI zur Verwendung mit TKG-Dienst-Clustern
- Stellen Sie mit TKr für vSphere 8.x einen TKG-Cluster bereit. Weitere Informationen finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl und https://docs.vmware.com/de/VMware-Tanzu-Kubernetes-releases/services/rn/vmware-tanzu-kubernetes-releases-release-notes/index.html in den Versionshinweisen zu TKR.

Installieren von Carvel imgpkg

Mit dem Carvel-Tool „imgpkg“ (https://carvel.dev/imgpkg/) können Sie die verfügbaren Versionen des Standardpaket-Repositorys durchsuchen. Das Repository ist öffentlich, sodass Sie sich nicht anmelden müssen.

Führen Sie die folgenden Schritte aus, um Carvel imgpkg zu installieren.

1 Installieren Sie imgpkg mithilfe des folgenden Befehls.

```
wget -O- https://carvel.dev/install.sh > install.sh
sudo bash install.sh
```

2 Überprüfen Sie die Installation.

```
imgpkg version
```

Beispielergebnis:

```
imgpkg version 0.42.1
```

3 Listen Sie die Repository-Versionen auf, indem Sie den folgenden Befehl ausführen.

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

Dieser Befehl gibt die verfügbaren Versionen des Standardpaket-Repositorys zurück.

```
Tags

Name
...
v2024.4.12
v2024.4.19
v2024.5.14
v2024.5.16

39 tags

Succeeded
```

Erstellen des Paket-Repositorys

Melden Sie sich beim TKG-Cluster an und erstellen Sie das Paket-Repository.

1 Melden Sie sich beim Cluster an.

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USER@vsphere.local --tanzu-kubernetes-cluster-name CLUSTER --tanzu-kubernetes-cluster-namespace VSPHERE-NS
```

2 Erstellen Sie das Paket-Repository.

```
tanzu package repository add standard-repo --url projects.registry.vmware.com/tkg/packages/standard/repo:v2024.5.16 -n tkg-system
```

Hinweis Ändern Sie die Versionszeichenfolge des Repositorys so, dass sie mit der Version des Ziel-Repositorys übereinstimmt.

3 Listen Sie die verfügbaren Pakete auf.

```
tanzu package available list -n tkg-system
```

Hinweis Nicht alle im Repository vorhandenen Pakete werden auf TKG-Clustern unterstützt. Sehen Sie hierzu die offizielle Liste der unterstützten Pakete: Standardpakete für TKG-Dienst-Cluster.

4 Listen Sie die für ein bestimmtes Paket verfügbaren Versionen auf.

Cert Manager

```
tanzu package available get cert-manager.tanzu.vmware.com -n tkg-system
```

Contour

```
tanzu package available get contour.tanzu.vmware.com -n tkg-system
```

Externes DNS

```
tanzu package available get external-dns.tanzu.vmware.com -n tkg-system
```

Fluent Bit

```
tanzu package available get fluent-bit.tanzu.vmware.com -n tkg-system
```

Grafana

```
tanzu package available get grafana.tanzu.vmware.com -n tkg-system
```

Prometheus

```
tanzu package available get prometheus.tanzu.vmware.com -n tkg-system
```

Installieren des Zertifikatmanagers

Befolgen Sie diese Anweisungen, um Cert Manager auf einem TKG-Dienst-Cluster zu installieren, auf dem TKr für vSphere 8.x ausgeführt wird.

Informationen zu Cert Manager

Cert Manager bietet eine Zertifikatsverwaltung für TKG-Dienstcluster. Cert Manager ist eine Voraussetzung für die meisten Standardpakete, einschließlich Contour, ExternalDNS, Prometheus und Harbor.

Voraussetzungen

Beachten Sie die folgenden Voraussetzungen.

- Allgemeine Anforderungen.\
- Erstellen des Paket-Repositorys

Installieren des Zertifikatmanagers

Führen Sie die folgenden Schritte aus, um Cert Manager zu installieren.

1 Listet die verfügbaren Cert Manager-Versionen auf.

```
tanzu package available get cert-manager.tanzu.vmware.com -n tkg-system
```

Hinweis In der Regel sollten Sie die neueste Version verwenden, es sei denn, Ihre Anforderungen weichen ab.

2 Erstellen Sie den Cert Manager-Namespace.

```
kubectl create ns cert-manager
```

3 Installieren Sie Cert Manager.

Passen Sie die Zielversion an Ihre Anforderungen an.

```
tanzu package install cert-manager -p cert-manager.tanzu.vmware.com -n cert-manager -v
1.12.2+vmware.2-tkg.2
```

4 Überprüfen Sie die Installation von Cert Manager.

```
tanzu package installed list -n cert-manager
```

```
tanzu package installed get -n cert-manager cert-manager
```

5 Überprüfen Sie den Cert Manager-Namespace auf im Rahmen der Installation des Pakets erstellte Ressourcen.

```
kubectl -n cert-manager get all
```

```
NAME                                       READY   STATUS    RESTARTS   AGE
pod/cert-manager-b5675b75f-flkjp           1/1     Running   0          6m14s
pod/cert-manager-cainjector-f8dc756cf-f7xsv   1/1     Running   0          6m14s
pod/cert-manager-webhook-6c888c8ddd-5xlnb     1/1     Running   0          6m14s

NAME                           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/cert-manager           ClusterIP   10.97.254.59     <none>        9402/TCP   6m14s
service/cert-manager-webhook   ClusterIP   10.105.225.156   <none>        443/TCP    6m14s

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cert-manager           1/1     1            1           6m14s
deployment.apps/cert-manager-cainjector   1/1     1            1           6m14s
deployment.apps/cert-manager-webhook      1/1     1            1           6m14s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/cert-manager-b5675b75f            1         1         1       6m14s
replicaset.apps/cert-manager-cainjector-f8dc756cf   1         1         1       6m14s
replicaset.apps/cert-manager-webhook-6c888c8ddd     1         1         1       6m14s
```

Beheben

Verwenden Sie die folgenden Befehle, um nach Fehlermeldungen zu suchen.

```
kubectl get pkgi -A
```

```
kubectl describe pkgi -n cert-manager cert-manage
```

Installieren von Contour mit Envoy

Befolgen Sie diese Anweisungen, um Contour mit Envoy auf einem TKG-Dienst-Cluster zu installieren, auf dem TKr für vSphere 8.x ausgeführt wird.

Voraussetzungen

Beachten Sie die folgenden Voraussetzungen.

- Allgemeine Anforderungen
- Referenz zum Contour-Paket
- Erstellen des Paket-Repositorys
- Installieren des Zertifikatmanagers

Erstellen von Contour-Datenwerten

Bereiten Sie die Installation von Contour vor, indem Sie die Datenwertdatei erstellen.

1 Listen Sie die verfügbaren Contour-Paketversionen auf.

```
tanzu package available get contour.tanzu.vmware.com -n tkg-system
```

Oder verwenden Sie dazu „kubectl“:

```
kubectl -n tkg-system get packages | grep contour
```

Hinweis In der Regel sollten Sie die neueste Version verwenden, es sei denn, Ihre Anforderungen weichen ab.

2 Generieren Sie die Datei `contour-default-values.yaml`.

```
tanzu package available get contour.tanzu.vmware.com/1.28.2+vmware.1-tkg.1 --default-values-file-output contour-data-values.yaml
```

Dabei gilt:

- *1.28.2+vmware.1-tkg.1* ist die Zielpaketversion
- *contour-data-values.yaml* ist der Name und Pfad der zu generierenden Datenwertdatei

3 Bearbeiten Sie die Datei `contour-data-values.yaml`.

Legen Sie den Envoy-Dienst auf `LoadBalancer` fest, damit von außerhalb des Clusters stammender Datenverkehr auf Kubernetes-Dienste zugreifen kann. Weitere Informationen finden Sie im folgenden Beispiel.

```
vi contour-data-values.yaml
```

```
---
infrastructure_provider: vsphere
namespace: tanzu-system-ingress
contour:
 configFileContents: {}
 useProxyProtocol: false
 replicas: 2
 pspNames: "vmware-system-restricted"
 logLevel: info
envoy:
 service:
   type: LoadBalancer
   annotations: {}
   externalTrafficPolicy: Cluster
   disableWait: false
 hostPorts:
   enable: true
   http: 80
   https: 443
 hostNetwork: false
```

```
 terminationGracePeriodSeconds: 300
 logLevel: info
certificates:
 duration: 8760h
 renewBefore: 360h
```

Installieren von Contour

Führen Sie diese Schritte aus, um Contour Ingress mit Envoy zu installieren.

1 Erstellen Sie einen eindeutigen Namespace für das Contour-Paket.

```
kubectl create ns tanzu-system-ingress
```

2 Installieren Sie Contour.

Passen Sie die Version an Ihre Anforderungen an.

```
tanzu package install contour -p contour.tanzu.vmware.com -v 1.28.2+vmware.1-tkg.1 --
values-file contour-data-values.yaml -n tanzu-system-ingress
```

3 Überprüfen Sie die Contour-Installation.

```
tanzu package installed list -n tanzu-system-ingress
```

4 Überprüfen Sie die Contour- und Envoy-Objekte.

```
kubectl -n tanzu-system-ingress get all
```

```
NAME                            READY   STATUS    RESTARTS   AGE
pod/contour-777bdddc69-fqnsp    1/1     Running   0          102s
pod/contour-777bdddc69-gs5xv    1/1     Running   0          102s
pod/envoy-d4jtt                 2/2     Running   0          102s
pod/envoy-g5h72                 2/2     Running   0          102s
pod/envoy-pjpzc                 2/2     Running   0          102s

NAME              TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)                      AGE
service/contour   ClusterIP      10.105.242.46   <none>
8001/TCP                     102s
service/envoy     LoadBalancer   10.103.245.57   10.197.154.69   80:32642/
TCP,443:30297/TCP   102s

NAME                    DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
daemonset.apps/envoy    3         3         3       3            3
<none>          102s

NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/contour     2/2     2            2           102s

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/contour-777bdddc69     2         2         2       102s
```

In diesem Beispiel hat der Envoy-Dienst die externe IP-Adresse 10.197.154.69. Diese IP-Adresse wird von dem CIDR-Bereich getrennt, der für **Arbeitslastnetzwerk > Ingress** angegeben ist. Eine neue Lastausgleichsdienst-Instanz wird für diese IP-Adresse erstellt. Die Mitglieder des Serverpools für diesen Lastausgleichsdienst sind die Envoy-Pods. Da die Envoy-Pods die IP-Adressen der Worker-Knoten übernehmen, auf denen sie ausgeführt werden, können Sie diese IP-Adressen anzeigen, indem Sie die Clusterknoten abfragen (`kubectl get nodes -o wide`).

Beheben

Lesen Sie bei Bedarf das folgende Thema.

- Referenz zum Contour-Paket.

Installieren von ExternalDNS

Befolgen Sie diese Anweisungen, um ExternalDNS auf einem TKG-Dienst-Cluster zu installieren, auf dem TKr für vSphere 8.x ausgeführt wird.

Informationen zu ExternalDNS

ExternalDNS ermöglicht die automatische Erstellung von DNS-Datensätzen für Kubernetes-Dienste mit einer Ingress-Komponente wie Contour mit Envoy. Das ExternalDNS-Paket wird mit den folgenden DNS-Anbietern validiert: AWS Route 53, Azure DNS und RFC2136-konformen DNS-Servern (z. B. BIND). Siehe auch Referenz zum ExternalDNS-Paket

Voraussetzungen

Beachten Sie die folgenden Voraussetzungen.

- Allgemeine Anforderungen.
- Erstellen des Paket-Repositorys.
- Installieren des Zertifikatmanagers.
- Installieren von Contour mit Envoy.

Erstellen von ExternalDNS-Datenwerten

Bereiten Sie die Installation von ExternalDNS vor, indem Sie die ExternalDNS-Datenwertdatei erstellen.

1 Listen Sie die im Repository verfügbaren ExternalDNS-Paketversionen auf.

```
tanzu package available get external-dns.tanzu.vmware.com -n tkg-system
```

Oder verwenden Sie dazu „kubectl“.

```
kubectl -n tkg-system get packages | grep external-dns
```

Hinweis In der Regel sollten Sie die neueste Version verwenden, es sei denn, Ihre Anforderungen weichen ab.

2 Generieren Sie die Datenwertdatei für das ExternalDNS-Paket.

```
tanzu package available get external-dns.tanzu.vmware.com/0.13.6+vmware.1-tkg.1 --default-values-file-output external-dns-data-values.yaml
```

Dabei gilt:

- *0.13.6+vmware.1-tkg.1* ist die Zielpaketversion
- *external-dns-data-values.yaml* ist der Name und Pfad der zu generierenden Datenwertdatei

3 Passen Sie die Datenwerte für Ihre Umgebung an.

Die Datenwerte unterscheiden sich je nach dem unterstützten DNS-Server, den Sie als Ziel verwenden. Beispiele finden Sie unter Referenz zum ExternalDNS-Paket.

4 Erstellen Sie bei Bedarf eine ConfigMap, die den DNS-Server definiert, mit dem das ExternalDNS-Paket verbunden wird.

Ein Beispiel finden Sie unter Referenz zum ExternalDNS-Paket.

Installieren von ExternalDNS

Führen Sie diese Schritte aus, um das ExternalDNS-Paket auf einem TKG-Cluster zu installieren.

1 Erstellen Sie einen Namespace für ExternalDNS.

```
kubectl create ns tanzu-system-service-discovery
```

2 Installieren Sie das ExternalDNS-Pakets mithilfe der Tanzu-CLI.

```
tanzu package install external-dns -p external-dns.tanzu.vmware.com -n tanzu-system-service-discovery -v 0.11.0+vmware.1-tkg.2 --values-file external-dns-data-values.yaml
```

3 Stellen Sie sicher, dass das Paket mithilfe der Tanzu CLI installiert wurde.

```
tanzu package installed list -n tanzu-system-service-discovery

NAME          PACKAGE-NAME                    PACKAGE-VERSION        STATUS
external-dns  external-dns.tanzu.vmware.com  0.11.0+vmware.1-tkg.2  Reconcile succeeded
```

```
kubectl -n tanzu-system-service-discovery get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/external-dns-77d947745-tcjz9   1/1     Running   0          63s

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
```

```
deployment.apps/external-dns   1/1     1            1           63s

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/external-dns-77d947745    1         1         1       63s
```

Referenz

Lesen Sie bei Bedarf die folgenden Themen.

- Referenz zum ExternalDNS-Paket.
- Installieren von ExternalDNS für die Diensterkennung.

Installieren von Fluent Bit

Befolgen Sie diese Anweisungen, um Fluent Bit auf einem TKG-Dienst-Cluster zu installieren, auf dem TKr für vSphere 8.x ausgeführt wird.

Voraussetzungen

Beachten Sie die folgenden Voraussetzungen.

- Allgemeine Anforderungen
- Erstellen des Paket-Repositorys
- Installieren des Zertifikatmanagers
- Ein unterstütztes Ziel für die Fluent Bit-Protokollweiterleitung. Weitere Informationen hierzu finden Sie unter Referenz zum Fluent Bit-Paket.

Erstellen von Fluent Bit-Datenwerten

Bereiten Sie die Installation von Fluent Bit vor, indem Sie die Datenwertdatei erstellen.

1 Listen Sie die verfügbaren Contour-Paketversionen auf.

```
tanzu package available get fluent-bit.tanzu.vmware.com -n tkg-system
```

Oder verwenden Sie dazu „kubectl“:

```
kubectl -n tkg-system get packages | grep fluent-bit
```

Hinweis In der Regel sollten Sie die neueste Version verwenden, es sei denn, Ihre Anforderungen weichen ab.

2 Generieren Sie die Datei `fluent-bit-data-values.yaml`.

```
tanzu package available get fluent-bit.tanzu.vmware.com/2.1.6+vmware.1-tkg.2 --default-values-file-output fluent-bit-data-values.yaml
```

3 Bearbeiten Sie die Datei `fluent-bit-data-values.yaml` und konfigurieren Sie die Werte.

Beispiele und eine Liste mit allen verfügbaren Parametern finden Sie unter Referenz zum Fluent Bit-Paket.

Installieren von Fluent Bit

Führen Sie diese Schritte aus, um das Fluent Bit-Paket zu installieren.

1 Erstellen Sie den Namespace für Fluent Bit.

```
kubectl create ns tanzu-system-logging
```

2 Installieren Sie Fluent Bit.

```
tanzu package install fluent-bit -p fluent-bit.tanzu.vmware.com -v 2.1.6+vmware.1-tkg.2
--values-file fluent-bit-data-values.yaml -n tanzu-system-logging
```

3 Überprüfen Sie die Fluent Bit-Installation.

```
tanzu package installed list -n tanzu-system-logging
```

```
tanzu package installed get fluent-bit -n tanzu-system-logging
```

4 Überprüfen Sie die Fluent Bit-Objekte.

```
kubectl -n tanzu-system-logging get all
```

Installieren von Prometheus mit Alertmanager

Befolgen Sie diese Anweisungen, um Prometheus mit Alertmanager auf einem TKG-Dienst-Cluster zu installieren, auf dem TKr für vSphere 8.x ausgeführt wird.

Voraussetzungen

Beachten Sie die folgenden Voraussetzungen.

- Allgemeine Anforderungen
- Erstellen des Paket-Repositorys.
- Installieren des Zertifikatmanagers.
- Installieren von Contour mit Envoy (erforderlich für den Zugriff auf das Prometheus-Dashboard).
- Prometheus-Paketreferenz

Erstellen von Prometheus-Datenwerten

Bereiten Sie die Installation von Prometheus vor, indem Sie die Datenwertdatei erstellen.

1 Rufen Sie die neueste Prometheus-Paketversion für Ihr Repository ab.

```
tanzu package available get prometheus.tanzu.vmware.com -n tkg-system
```

Oder verwenden Sie dazu „kubectl“.

```
kubectl -n tkg-system get packages | grep prometheus
```

Hinweis In der Regel sollten Sie die neueste Version verwenden, es sei denn, Ihre Anforderungen weichen ab.

2 Generieren Sie die Datei `prometheus-data-values.yaml`.

```
tanzu package available get prometheus.tanzu.vmware.com/2.45.0+vmware.1-tkg.2 --default-values-file-output prometheus-data-values.yaml
```

Dabei gilt:

- *2.45.0+vmware.1-tkg.2* ist die Zielpaketversion
- *prometheus-data-values.yaml* ist der Name und Pfad der zu generierenden Datenwertdatei

3 Bearbeiten Sie die Datei `prometheus-data-values.yaml`, und konfigurieren Sie die folgenden Werte, die für den Zugriff auf das Prometheus-Dashboard erforderlich sind. Unter Prometheus-Paketreferenz finden Sie eine Beispieldatei mit Datenwerten und eine vollständige Liste der Konfigurationsparameter.

Parameter	Beschreibung
`ingress.tlsCertificate.tls.crt`	Für den eingehenden Datenverkehr wird ein selbstsigniertes TLS-Zertifikat generiert. Dieses können Sie optional außer Kraft setzen und Ihr eigenes Zertifikat angeben.
`ingress.tlsCertificate.tls.key`	Für den eingehenden Datenverkehr wird ein selbstsignierter privater TLS-Schlüssel generiert. Dieses können Sie optional außer Kraft setzen und Ihr eigenes Zertifikat angeben.
`ingress.enabled`	Legen Sie den Wert auf `true` fest (Standard ist „false“).
`ingress.virtual_host_fqdn`	Legen Sie den Wert auf `prometheus.<your.domain>` fest (Standard ist `prometheus.system.tanzu`).
`alertmanager.pvc.storageClassName`	Geben Sie den Namen der vSphere-Speicherrichtlinie ein.
`prometheus.pvc.storageClassName`	Geben Sie den Namen der vSphere-Speicherrichtlinie ein.

Installieren von Prometheus

Führen Sie diese Schritte aus, um das Prometheus-Paket zu installieren.

1 Erstellen Sie den Namespace.

```
kubectl create ns tanzu-system-monitoring
```

2 Installieren Sie Prometheus.

```
tanzu package install prometheus -p prometheus.tanzu.vmware.com -v 2.45.0+vmware.1-tkg.2
--values-file prometheus-data-values.yaml -n tanzu-system-monitoring
```

3 Überprüfen Sie die Prometheus-Installation.

```
tanzu package installed list -n tanzu-system-monitoring
```

```
tanzu package installed get prometheus -n tanzu-system-monitoring
```

4 Überprüfen Sie die Prometheus- und Alertmanager-Objekte.

```
kubectl -n tanzu-system-monitoring get all
```

```
kubectl -n tanzu-system-monitoring get pvc

NAME                 STATUS   VOLUME                                     CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
alertmanager         Bound    pvc-a53f7091-9823-4b70-a9b4-c3d7a1e27a4b   2Gi
RWO             k8s-policy     2m30s
prometheus-server    Bound    pvc-41745d1d-9401-41d7-b44d-ba430ecc5cda   20Gi
RWO             k8s-policy     2m30s
```

Fehlerbehebung bei der Prometheus-Installation

Wenn beim `tanzu package install prometheus`-Vorgang der Fehler „Failed to get final advertise address: No private IP address found, and explicit IP not provided“ (Fehler beim Abrufen der endgültigen Ankündigungsadresse: Keine private IP-Adresse gefunden und explizite IP nicht angegeben) zurückgegeben wird, wenden Sie ein Paket-Overlay an, um die Alertmanager-Komponente neu zu konfigurieren.

1 Erstellen Sie die Datei „`overlay-alertmanager.yaml`“.

```
---
#@ load("@ytt:overlay", "overlay")

#@overlay/match by=overlay.and_op(overlay.subset({"kind": "Deployment"}),
overlay.subset({"metadata": {"name": "alertmanager"}}))
---
spec:
  template:
    spec:
      containers:
        #@overlay/match by="name",expects="0+"
        - name: alertmanager
          args:
            - --cluster.listen-address=
```

2 Erstellen Sie mithilfe von Kubectl einen geheimen Schlüssel aus der `overlay-alertmanager.yaml`-Datei.

```
kubectl create secret generic alertmanager-overlay -n tkg-system -o yaml --dry-run=client
--from-file=overlay-alertmanager.yaml | kubectl apply -f -
```

3 Verwenden Sie Kubectl, um das Prometheus-Paket mit dem geheimen Overlay-Schlüssel zu versehen.

```
kubectl annotate PackageInstall prometheus -n tkg-system ext.packaging.carvel.dev/ytt-
paths-from-secret-name.1=alertmanager-overlay
```

4 Führen Sie den Installationsbefehl erneut aus.

```
tanzu package install prometheus -p prometheus.tanzu.vmware.com -v 2.37.0+vmware.3-tkg.1
--values-file prometheus-data-values.yaml -n tanzu-system-monitoring
```

Zugreifen auf das Prometheus-Dashboard

Führen Sie nach der Installation von Prometheus die folgenden Schritte aus, um auf das Prometheus-Dashboard zuzugreifen.

1 Stellen Sie sicher, dass der `ingress`-Abschnitt der `prometheus-data-values.yaml`-Datei mit allen erforderlichen Feldern gefüllt ist.

```
ingress:
  enabled: true
  virtual_host_fqdn: "prometheus.system.tanzu"
  prometheus_prefix: "/"
  alertmanager_prefix: "/alertmanager/"
  prometheusServicePort: 80
  alertmanagerServicePort: 80
  #! [Optional] The certificate for the ingress if you want to use your own TLS
certificate.
  #! We will issue the certificate by cert-manager when it's empty.
  tlsCertificate:
    #! [Required] the certificate
    tls.crt:
    #! [Required] the private key
    tls.key:
    #! [Optional] the CA certificate
    ca.crt:
```

2 Rufen Sie die öffentliche (externe) IP-Adresse des Contour mit Envoy-Lastausgleichsdiensts ab.

Weitere Informationen hierzu finden Sie unter Installieren von Contour mit Envoy.

3 Erstellen Sie einen DNS-Datensatz, der den von Ihnen verwendeten Prometheus-FQDN (standardmäßig `prometheus.system.tanzu`) der IP-Adresse des Envoy-Lastausgleichsdiensts zuordnet.

4 Greifen Sie auf das Prometheus-Dashboard zu, indem Sie mit einem Browser zum Prometheus-FQDN navigieren.

Installieren von Grafana

Befolgen Sie diese Anweisungen, um Grafana auf einem TKG-Dienst-Cluster zu installieren, auf dem TKr für vSphere 8.x ausgeführt wird.

Voraussetzungen

Beachten Sie die folgenden Voraussetzungen.

- Allgemeine Anforderungen.
- Erstellen des Paket-Repositorys
- Installieren des Zertifikatmanagers
- Installieren von Contour mit Envoy
- Installieren von Prometheus mit Alertmanager
- Referenz zum Grafana-Paket

Erstellen von Grafana-Datenwerten

Bereiten Sie die Installation von Grafana vor, indem Sie die Datenwertdatei erstellen.

1 Rufen Sie die neueste Prometheus-Paketversion für Ihr Repository ab.

```
tanzu package available get grafana.tanzu.vmware.com -n tkg-system
```

Oder verwenden Sie dazu „kubectl“.

```
kubectl -n tkg-system get packages | grep grafana
```

Hinweis In der Regel sollten Sie die neueste Version verwenden, es sei denn, Ihre Anforderungen weichen ab.

2 Generieren Sie die Datei `prometheus-data-values.yaml`.

```
tanzu package available get grafana.tanzu.vmware.com/10.0.1+vmware.1-tkg.2 --default-
values-file-output grafana-data-values.yaml
```

Dabei gilt:

- *10.0.1+vmware.1-tkg.2* ist die Zielpaketversion
- *grafana-data-values.yaml* ist der Name und Pfad der zu generierenden Datenwertdatei

3 Bearbeiten Sie die `gafana-data-values.yaml`-Datei und aktualisieren Sie die Werte.

Fügen Sie den `ingress.pvc: storageClassName` und den zugehörigen Wert hinzu. Dabei handelt es sich um den Namen der vSphere-Speicherklasse, auf die der TKG-Cluster zugreifen kann.

Um einen häufigen Fehler zu vermeiden, entfernen Sie den geheimen Schlüssel aus der Datei mit den Datenwerten und erstellen Sie ihn manuell. Weitere Informationen hierzu finden Sie unter Fehlerbehebung bei der Grafana-Installation.

Hier sehen Sie eine minimale `gafana-data-values.yaml`-Datei, in der das Speicherklassenfeld hinzugefügt und der geheime Schlüssel entfernt wurde. Unter Referenz zum Grafana-Paket finden Sie ein zusätzliches Beispiel und eine vollständige Liste der Parameter.

```
grafana:
  deployment:
    replicas: 1
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 2Gi
  service:
    port: 80
    targetPort: 3000
    type: LoadBalancer
ingress:
  enabled: true
  prefix: /
  servicePort: 80
  virtual_host_fqdn: grafana.system.tanzu
  pvc:
    storageClassName: vSphere-storage-profile
namespace: grafana
```

Installieren von Grafana

Führen Sie diese Schritte aus, um das Grafana-Paket zu installieren.

1 Erstellen Sie den Namespace für Grafana.

```
kubectl create ns tanzu-system-dashboards
```

2 Installieren Sie das Grafana-Paket.

```
tanzu package install grafana -p grafana.tanzu.vmware.com -v 10.0.1+vmware.1-tkg.2 --
values-file grafana-data-values.yaml -n tanzu-system-dashboards
```

3 Überprüfen Sie die Grafana-Installation.

```
tanzu package installed list -n tanzu-system-dashboards
```

```
tanzu package installed get grafana -n tanzu-system-dashboards
```

4 Überprüfen Sie die Grafana-Objekte.

```
kubectl -n tanzu-system-dashboards get all
```

5 Überprüfen Sie die Beanspruchung eines dauerhaften Volumes von Grafana.

```
kubectl -n tanzu-system-dashboards get pvc
```

Fehlerbehebung bei der Grafana-Installation

Damit der Fehler „Geheimer Schlüssel wird bei der Installation von Grafana aus der YAML-Standarddatei nicht erstellt“ vermieden wird, entfernen Sie `grafana.secret.*` aus der Datei *grafana-data-values.yaml*, und erstellen Sie den geheimen Schlüssel wie folgt manuell. Stellen Sie dann das Grafana-Paket erneut bereit.

```
kubectl create secret generic grafana -n tanzu-system-dashboards  --from-literal=admin=admin
```

Installieren der Harbor-Registrierung

Befolgen Sie diese Anweisungen, um die Harbor-Containerregistrierung auf einem TKG-Dienst-Cluster zu installieren, auf dem TKr für vSphere 8.x ausgeführt wird.

Voraussetzungen

Beachten Sie die folgenden Voraussetzungen.

- Allgemeine Anforderungen
- Erstellen des Paket-Repositorys
- Installieren des Zertifikatmanagers
- Installieren von Contour mit Envoy
- Referenz zum Harbor-Paket

Erstellen von Harbor-Datenwerten

Bereiten Sie die Installation von Harbor vor, indem Sie die Datenwertdatei erstellen.

1 Rufen Sie die neueste Harbor-Paketversion für Ihr Repository ab.

```
tanzu package available get harbor.tanzu.vmware.com -n tkg-system
```

Oder verwenden Sie dazu „kubectl“.

```
kubectl -n tkg-system get packages | grep harbor
```

Hinweis In der Regel sollten Sie die neueste Version verwenden, es sei denn, Ihre Anforderungen weichen ab.

2 Generieren Sie die Datei `harbor-data-values.yaml`.

```
tanzu package available get harbor.tanzu.vmware.com/2.9.1+vmware.1-tkg.1 --default-values-file-output harbor-data-values.yaml
```

Dabei gilt:

- *2.9.1+vmware.1-tkg.1* ist die Zielpaketversion
- *harbor-data-values.yaml* ist der Name und Pfad der zu generierenden Datenwertdatei

3 Bearbeiten Sie die `harbor-data-values.yaml`-Datei und aktualisieren Sie die Werte für die folgenden Parameter.

Konfigurieren Sie nach Bedarf zusätzliche Parameter. Weitere Informationen hierzu finden Sie unter Referenz zum Harbor-Paket.

Bereich	Beschreibung
`hostname`	Der FQDN für den Zugriff auf die Harbor-Verwaltungskonsole und den Registrierungsdienst. Ersetzen Sie "yourdomain.com" durch einen eindeutigen Hostnamen.
`harborAdminPassword`	Ändern Sie das Kennwort in ein starkes und eindeutiges Kennwort (es kann auch nach der Installation in der Benutzeroberfläche geändert werden).
`persistence.persistentVolumeClaim.database.storageClass:`	Geben Sie den Namen der vSphere-Speicherrichtlinie für den vSphere-Namespace ein.
`persistence.persistentVolumeClaim.jobservice.storageClass:`	Geben Sie den Namen der vSphere-Speicherrichtlinie für den vSphere-Namespace ein.
`persistence.persistentVolumeClaim.redis.storageClass:`	Geben Sie den Namen der vSphere-Speicherrichtlinie für den vSphere-Namespace ein.
`persistence.persistentVolumeClaim.registry.storageClass:`	Geben Sie den Namen der vSphere-Speicherrichtlinie für den vSphere-Namespace ein.
`persistence.persistentVolumeClaim.trivy.storageClass:`	Geben Sie den Namen der vSphere-Speicherrichtlinie für den vSphere-Namespace ein.
`tlsCertificate.tlsSecretLabels:`	{"managed-by": "vmware-vRegistry"}

Installieren von Harbor

Führen Sie die folgenden Schritte aus, um die Harbor-Registrierung zu installieren.

1 Erstellen Sie den Namespace für Harbor.

```
kubectl create ns tanzu-system-registry
```

2 Installieren Sie Harbor.

```
tanzu package install harbor --package harbor.tanzu.vmware.com --version 2.9.1+vmware.1-tkg.1 --values-file harbor-data-values.yaml --namespace tanzu-system-registry
```

3 Überprüfen Sie die Harbor-Installation.

```
tanzu package installed get harbor --namespace tanzu-system-registry
```

Konfigurieren von DNS für Harbor mithilfe eines Envoy-Diensts vom Typ „LoadBalancer“

Wenn der erforderliche Contour mit Envoy-Dienst über einen LoadBalancer verfügbar gemacht wird, rufen Sie die externe IP-Adresse des Lastausgleichsdiensts ab und erstellen Sie DNS-Datensätze für die Harbor-FQDNs.

1 Rufen Sie die `External-IP`-Adresse für den Envoy-Dienst vom Typ LoadBalancer ab.

```
kubectl get service envoy -n tanzu-system-ingress
```

Die `External-IP`-Adresse sollte angezeigt werden, z. B.:

```
NAME    TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
envoy   LoadBalancer   10.99.25.220   10.195.141.17   80:30437/TCP,443:30589/TCP   3h27m
```

Alternativ können Sie die `External-IP`-Adresse mit dem folgenden Befehl abrufen.

```
kubectl get svc envoy -n tanzu-system-ingress -o
jsonpath='{.status.loadBalancer.ingress[0]}'
```

2 Um die Installation der Harbor-Erweiterung zu überprüfen, aktualisieren Sie Ihre lokale `/etc/hosts`-Datei mit den Harbor- und Notariats-FQDNs, die der `External-IP`-Adresse des Lastausgleichsdiensts zugeordnet sind. Beispiel:

```
127.0.0.1 localhost
127.0.1.1 ubuntu
#TKG Harbor with Envoy Load Balancer IP
10.195.141.17 core.harbor.domain
10.195.141.17 core.notary.harbor.domain
```

3 Um die Installation der Harbor-Erweiterung zu überprüfen, melden Sie sich bei Harbor an.

4 Erstellen Sie zwei CNAME-Datensätze auf einem DNS-Server, die die `External-IP`-Adresse des Lastausgleichsdiensts für den Envoy-Dienst dem Harbor-FQDN und dem Notariats-FQDN zuordnen.

5 Installieren Sie die externe DNS-Erweiterung.

Konfigurieren von DNS für Harbor mithilfe eines Envoy-Diensts vom Typ „NodePort“

Wenn der erforderliche Contour- > Envoy-Dienst über einen NodePort verfügbar gemacht wird, rufen Sie die IP-Adresse der virtuellen Maschine eines Worker-Knotens ab und erstellen Sie DNS-Datensätze für die Harbor-FQDNs.

Hinweis Um NodePort zu verwenden, müssen Sie den korrekten `port.https`-Wert in der `harbor-data-values.yaml`-Datei angegeben haben.

1 Wechseln Sie den Kontext zum vSphere-Namespace, in dem der Cluster bereitgestellt wird.

```
kubectl config use-context VSPHERE-NAMESPACE
```

2 Listet die Knoten im Cluster auf.

```
kubectl get virtualmachines
```

3 Wählen Sie einen der Worker-Knoten aus und beschreiben Sie ihn mit dem folgenden Befehl.

```
kubectl describe virtualmachines tkg2-cluster-X-workers-9twdr-59bc54dc97-kt4cm
```

4 Suchen Sie die IP-Adresse der virtuellen Maschine, z. B. `Vm Ip: 10.115.22.43`.

5 Um die Installation der Harbor-Erweiterung zu überprüfen, aktualisieren Sie Ihre lokale `/etc/hosts`-Datei mit den Harbor- und Notariats-FQDNs, die der IP-Adresse des Worker-Knotens zugeordnet sind. Beispiel:

```
127.0.0.1 localhost
127.0.1.1 ubuntu
#TKG Harbor with Envoy NodePort
10.115.22.43 core.harbor.domain
10.115.22.43 core.notary.harbor.domain
```

6 Um die Installation der Harbor-Erweiterung zu überprüfen, melden Sie sich bei Harbor an.

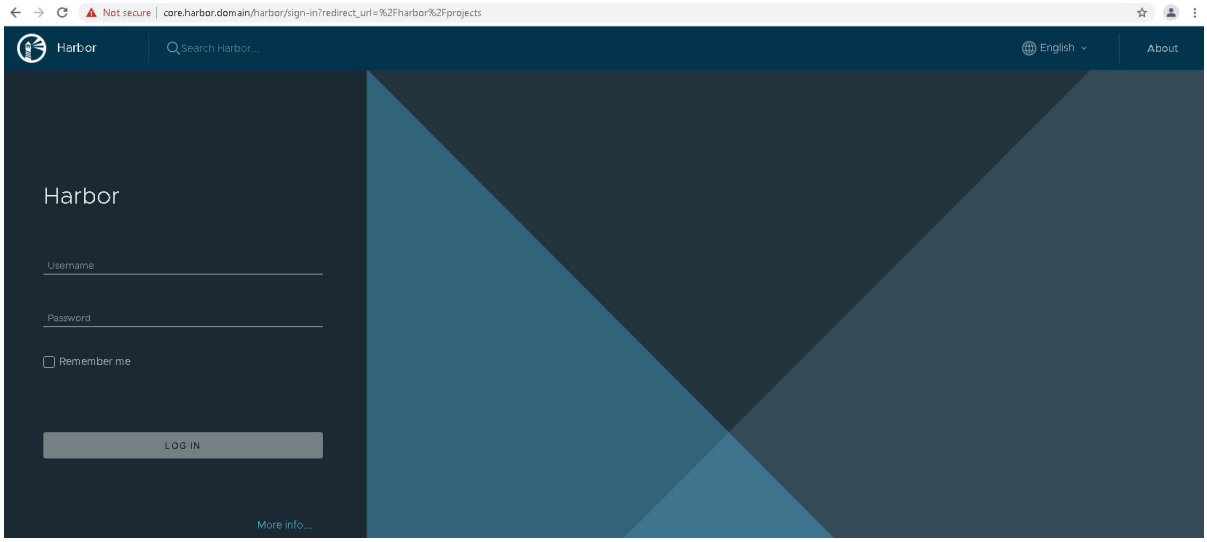
7 Erstellen Sie zwei CNAME-Datensätze auf einem DNS-Server, die die IP-Adresse des Worker-Knotens für den Envoy-Dienst dem Harbor-FQDN und dem Notariats-FQDN zuordnen.

8 Installieren Sie die externe DNS-Erweiterung.

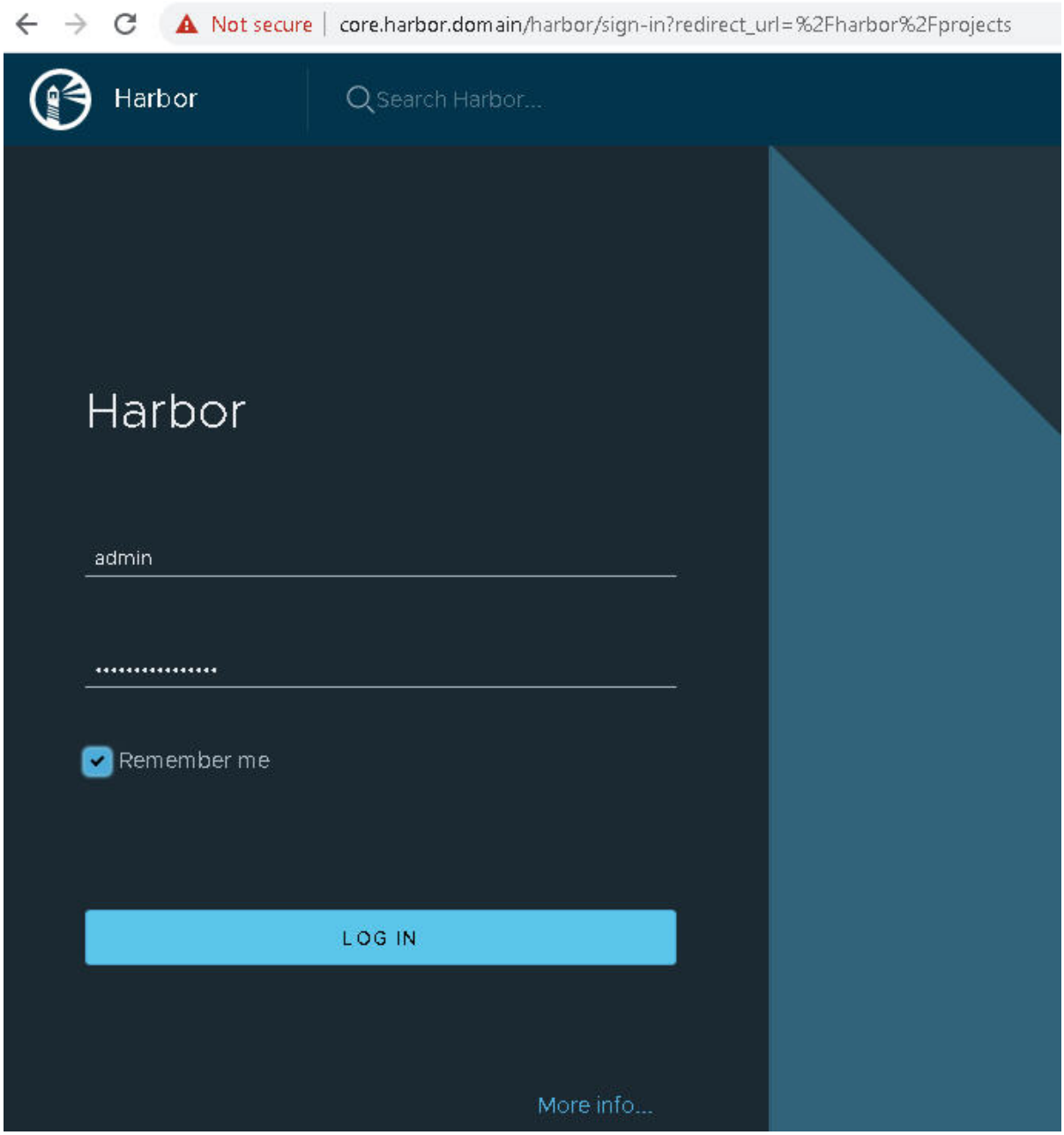
Anmelden bei der Harbor-Webschnittstelle

Sobald Harbor installiert und konfiguriert ist, melden Sie sich an und verwenden Sie es.

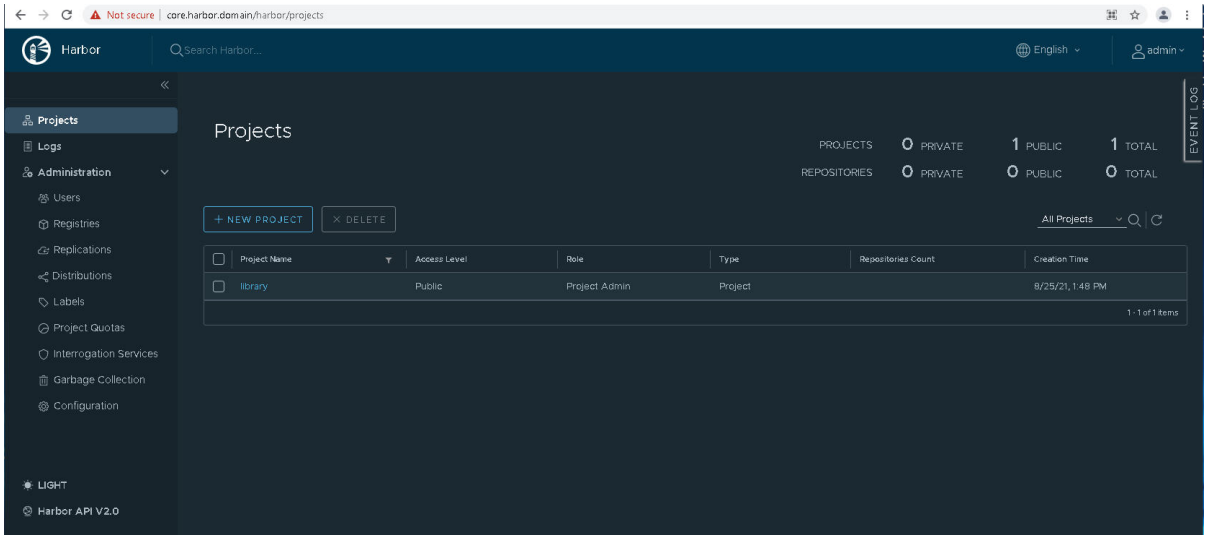
1 Greifen Sie unter `https://core.harbor.domain` oder dem von Ihnen verwendeten Hostnamen auf die Webschnittstelle der Harbor-Registrierung zu.



2 Melden Sie sich bei Harbor mit dem Benutzernamen **`admin`** und dem generierten Kennwort an, das Sie in die `harbor-data-values.yaml`-Datei eingegeben haben.



3 Überprüfen Sie, ob Sie über die Harbor-Benutzeroberfläche auf den Host zugreifen können.



4 Rufen Sie das Harbor-CA-Zertifikat ab.

   Klicken Sie in der Harbor-Benutzeroberfläche auf **Projekte > Bibliothek**, oder erstellen Sie ein **Neues Projekt**.

   Klicken Sie auf **Registrierungszertifikat** und laden Sie das Harbor-CA-Zertifikat (`ca.crt`) herunter.

5 Fügen Sie das Harbor-CA-Zertifikat zum Trust Store des Docker-Clients hinzu, damit Sie Container-Images in die Harbor-Registrierung verschieben und von dort abrufen können. Weitere Informationen finden Sie unter Kapitel 14 Verwenden privater Registrierungen mit TKG-Dienstclustern.

6 Weitere Informationen zur Verwendung von Harbor finden Sie in der Harbor-Dokumentation.

Standardpaketreferenz

Dieser Abschnitt enthält Referenzinformationen für die Standardpakete, die in TKG-Dienstclustern installiert werden können.

Referenz zum Contour-Paket

Dieses Thema enthält Referenzinformationen für das Contour mit Envoy-Paket.

Informationen zu Contour und Envory

Contour (https://projectcontour.io/) ist ein Kubernetes-Ingress-Controller, der den Envoy-Reverse-HTTP Proxy umfasst. Contour mit Envoy wird häufig mit anderen Paketen wie ExternalDNS, Prometheus und Harbor verwendet.

Informationen zum Installieren des Contour-Pakets auf einem TKG-Cluster finden Sie in den folgenden Themen:

- Installieren von Contour mit Envoy

- #unique_173

Contour-Komponenten

Das Contour-Paket umfasst den Contour-Ingress-Controller und den Envoy-Reverse HTTP-Proxy. Diese Komponenten werden als Container installiert. Die Container werden aus der öffentlichen Registrierung abgerufen, die im Paket-Repository angegeben ist.

Container	Ressourcentyp	Replikate	Beschreibung
Envoy	DaemonSet	3	Hochleistungs-Reverse-Proxy
Contour	Bereitstellung	2	Verwaltungs- und Konfigurationsserver für Envoy

Contour-Datenwerte

Nachstehend finden Sie ein Beispiel `contour-data-values.yaml`.

Die einzige Anpassung besteht darin, dass der Envoy-Dienst vom Typ LoadBalancer ist (der Standard ist NodePort). Dies bedeutet, dass der Envoy-Dienst von außerhalb des Clusters für den Ingress zugänglich ist.

```
infrastructure_provider: vsphere
namespace: tanzu-system-ingress
contour:
 configFileContents: {}
 useProxyProtocol: false
 replicas: 2
 pspNames: "vmware-system-restricted"
 logLevel: info
envoy:
 service:
   type: LoadBalancer
   annotations: {}
   nodePorts:
     http: null
     https: null
   externalTrafficPolicy: Cluster
   disableWait: false
 hostPorts:
   enable: true
   http: 80
   https: 443
 hostNetwork: false
 terminationGracePeriodSeconds: 300
 logLevel: info
 pspNames: null
certificates:
 duration: 8760h
 renewBefore: 360h
```

Contour-Konfiguration

Die Contour-Paket-Konfigurationswerte werden in `contour-data-values.yaml` festgelegt. In der Tabelle sind die verfügbaren Parameter aufgeführt und beschrieben.

Tabelle 11-2. Contour Ingress-Konfigurationsparameter

Parameter	Beschreibung	Typ	Standard
infrastructure_provider	Infrastrukturanbieter Unterstützte Werte: vsphere, aws, azure	string	Obligatorischer Parameter
contour.namespace	Namespace, in dem Contour bereitgestellt wird	string	tanzu-system-ingress
contour.config.requestTimeout	Zeitüberschreitung für die Clientanforderung, die an Envoy übermittelt werden soll	time.Duration	0 s (Weitere Informationen finden Sie im folgenden Abschnitt)
contour.config.server.xdsServerType	Zu verwendender XDS-Servertyp: Unterstützte Werte: "contour" oder "envoy"	string	Null
contour.config.tls.minimumProtocolVersion	Minimale TLS-Version, die Contour aushandelt	string	1,1
contour.config.tls.fallbackCertificate.name	Name des geheimen Schlüssels, der das Fallback-Zertifikat für Anforderungen enthält, die nicht mit dem für einen vhost definierten SNI übereinstimmen	string	Null
contour.config.tls.fallbackCertificate.namespace	Namespace des geheimen Schlüssels, der das Fallback-Zertifikat enthält	string	Null
contour.config.tls.envoyClientCertificate.name	Name des geheimen Schlüssels, der als Clientzertifikat verwendet werden soll, privater Schlüssel für die TLS-Verbindung zum Backend-Dienst	string	Null
contour.config.tls.envoyClientCertificate.namespace	Namespace des geheimen Schlüssels, der als Clientzertifikat verwendet werden soll, privater Schlüssel für die TLS-Verbindung zum Backend-Dienst	string	Null

Tabelle 11-2. Contour Ingress-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
contour.config.leaderelection.configmapName	Name der Konfigurationszuordnung, die für die Contour-Leaderelection verwendet werden soll	string	leader-elect
contour.config.leaderelection.configmapNamespace	Namespace der Contour-Leaderelection-Configmap	string	tanzu-system-ingress
contour.config.disablePermitInsecure	Deaktiviert das Feld "ingressroute permitInsecure"	Boolean	false
contour.config.accesslogFormat	Zugriffsprotokollformat	string	Envoy
contour.config.jsonFields	Felder, die protokolliert werden	Array von Zeichenfolgen	Envoy-Paketdokument
contour.config.useProxyProtocol	https://projectcontour.io/guides/proxy-proto/	Boolean	false
contour.config.defaultHTTPVersions	HTTP-Versionen, mit denen Contour Envoy programmieren soll, damit sie von Envoy unterstützt werden	Array von Zeichenfolgen	"HTTP/1.1 HTTP2"
contour.config.timeouts.requestTimeout	Zeitüberschreitung für eine gesamte Anforderung	time.Duration	Null (Zeitüberschreitung ist deaktiviert)
config.timeouts.connectionIdleTimeout	Wartezeit bis zum Beenden einer Leerlaufverbindung	time.Duration	60 s
contour.config.timeouts.streamIdleTimeout	Wartezeit bis zum Beenden einer Anforderung oder eines Streams ohne Aktivität	time.Duration	5 m
contour.config.timeouts.maxConnectionDuration	Wartezeit bis zum Beenden einer Verbindung unabhängig von der Aktivität	time.Duration	Null (Zeitüberschreitung ist deaktiviert)
contour.config.timeouts.ConnectionShutdownGracePeriod	Wartezeit zwischen dem Senden eines ersten und eines letzten GOAWAY	time.Duration	5 s
contour.config.cluster.dnsLookupFamily	dns-lookup-family, die für Upstream-Anforderungen an Dienste vom Typ externalName von einer HTTPProxy-Route verwendet werden soll	string	Null (unterstützte Werte: auto, v4, v6)
contour.config.debug	Fehlerbehebung (Debuggen) durch Contour aktivieren	Boolean	false

Tabelle 11-2. Contour Ingress-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
contour.config.ingressStatusAddress	Die Adresse, die für den Status jeder Ingress-Ressource festgelegt werden soll	string	Null
contour.certificate.duration	Gültigkeit des Contour-Zertifikats	time.Duration	8760 h
contour.certificate.renewBefore	Dauer, bis das Contour-Zertifikat erneuert werden sollte	time.Duration	360 h
contour.deployment.replicas	Anzahl an Contour-Replikas	integer	2
contour.image.repository	Speicherort des Repositorys mit dem Contour-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg
contour.image.name	Name des Contour-Images	string	Contour
contour.image.tag	Contour-Image-Tag Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Contour-Version durchführen.	string	v1.11.0_vmware.1
contour.image.pullPolicy	Pull-Richtlinie für das Contour-Image	string	IfNotPresent
envoy.image.repository	Speicherort des Repositorys mit dem Envoy-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg
envoy.image.name	Name des Envoy-Images	string	Envoy
envoy.image.tag	Envoy-Image-Tag. Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Envoy-Version durchführen.	string	`v1.17.3_vmware.1`

Tabelle 11-2. Contour Ingress-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
envoy.image.pullPolicy	Pull-Richtlinie für das Envoy-Image	string	IfNotPresent
envoy.hostPort.enable	Flag, um Envoy-Ports auf dem Host verfügbar zu machen	Boolean	true
envoy.hostPort.http	Envoy HTTP-Host-Port	integer	80
envoy.hostPort.https	Envoy HTTPS-Host-Port	integer	443
envoy.service.type	Diensttyp zur Verfügbarmachung von von Envoy. Unterstützte Werte: ClusterIP, NodePort, LoadBalancer	string	Obligatorischer Parameter für vSphere: NodePort oder LoadBalancer, AWS: LoadBalancer, Azure: LoadBalancer
envoy.service.annotations	Envoy-Dienstanmerkungen	Zuordnung (key-values)	Leere Zuordnung
envoy.service.externalTrafficPolicy	Externe Datenverkehrsrichtlinie des Envoy-Dienstes. Unterstützte Werte: Lokal, Cluster	string	Cluster
envoy.service.nodePort.http	Gewünschter nodePort für den Dienst des Typs NodePort, der für HTTP-Anforderungen verwendet wird	integer	Null – Kubernetes weist einen dynamischen Knotenport zu
envoy.service.nodePort.https	Gewünschter nodePort für den Dienst vom Typ NodePort, der für HTTPS-Anforderungen verwendet wird	integer	Null – Kubernetes weist einen dynamischen Knotenport zu
envoy.deployment.hostNetwork	Envoy auf hostNetwork ausführen	Boolean	false
envoy.service.aws.LBType	AWS-LB-Typ, der für die Offenlegung des Envoy Service verwendet werden soll. Unterstützte Werte: classic, nlb	string	classic
envoy.loglevel	Für Envoy zu verwendende Protokollebene	string	Info

Routen-Zeitüberschreitung für Datei-Downloads

Der Parameter `contour.config.requestTimeout` definiert die Zeitüberschreitungsdauer der Contour-Route. Der Standardwert ist `0s`. Wenn Sie Contour für die Dateiübertragung verwenden, müssen Sie diesen Wert gegebenenfalls anpassen.

Gemäß der Contour-Dokumentation weist ein Zeitüberschreitungswert von `0s` Contour an, die Envoy-Zeitüberschreitung zu verwenden. Gemäß der Envoy-Dokumentation weist Envoy eine Standardzeitüberschreitung von 15 Sekunden auf. Des Weiteren geht Envoy davon aus, dass der gesamte Anforderung-Antwort-Vorgang innerhalb des Zeitüberschreitungsintervalls abgeschlossen wird.

Dies bedeutet, dass die Dateiübertragung bei der standardmäßigen Contour-Zeitüberschreitungseinstellung von `0s` innerhalb von 15 Sekunden abgeschlossen sein muss. Für große Dateiübertragungen reicht die Zeit möglicherweise nicht aus. Um die Envoy-Standardzeitüberschreitung zu deaktivieren, legen Sie den Wert `contour.config.requestTimeout` auf `0` fest.

Referenz zum ExternalDNS-Paket

Dieses Thema enthält Referenzinformationen für das ExternalDNS-Paket.

Informationen zu ExternalDNS

ExternalDNS synchronisiert bereitgestellte Kubernetes-Dienste und -Ingresses mit DNS-Anbietern.

Informationen zur Installation von ExternalDNS auf einem TKG-Cluster finden Sie in den im Folgenden aufgeführten Themen.

- TKr für vSphere 8x: Installieren von ExternalDNS
- TKr für vSphere 7.x: Installieren von ExternalDNS

ExternalDNS-Komponenten

Das ExternalDNS-Paket installiert den in der Tabelle aufgeführten Container. Das Paket ruft den Container aus der öffentlichen Registrierung ab, die im Paket-Repository angegeben ist.

Container	Ressourcentyp	Replikate	Beschreibung
ExternalDNS	DaemonSet	6	Kubernetes-Dienste für DNS-Lookup verfügbar machen

ExternalDNS-Datenwerte

Die ExternalDNS-Datenwertedatei wird verwendet, um die ExternalDNS-Komponente mit einem unterstützten DNS-Anbieter zu verbinden. Das ExternalDNS-Paket wird mit den folgenden DNS-Anbietern validiert: AWS (Route 53), Azure DNS und RFC2136-konformen DNS-Servern (z. B. BIND).

Das folgende Beispiel kann für einen RFC2136-konformen DNS-Anbieter (z. B. BIND) verwendet werden.

```
---
# Namespace in which to deploy ExternalDNS pods
namespace: tanzu-system-service-discovery
# Deployment-related configuration
```

```
deployment:
args:
   - --registry=txt
   - --txt-owner-id=k8s
   - --txt-prefix=external-dns- #! Disambiguates TXT records from CNAME records
   - --provider=rfc2136
   - --rfc2136-host=IP-ADDRESS #! Replace with IP of RFC2136-compatible DNS server, such as
192.168.0.1
   - --rfc2136-port=53
   - --rfc2136-zone=DNS-ZONE #! Replace with zone where services are deployed, such as my-
zone.example.org
   - --rfc2136-tsig-secret=TSIG-SECRET #! Replace with TSIG key secret authorized to update
DNS server
   - --rfc2136-tsig-secret-alg=hmac-sha256
   - --rfc2136-tsig-keyname=TSIG-KEY-NAME #! Replace with TSIG key name, such as externaldns-
key
   - --rfc2136-tsig-axfr
   - --source=service
   - --source=ingress
   - --source=contour-httpproxy #! Enables Contour HTTPProxy object support
   - --domain-filter=DOMAIN #! Zone where services are deployed, such as my-zone.example.org
```

Das folgende Beispiel kann für einen AWS DNS-Anbieter (Route 53) verwendet werden.

```
---
    namespace: service-discovery
    dns:
      pspNames: "vmware-system-restricted"
      deployment:
        args:
        - --source=service
        - --source=ingress
        - --source=contour-httpproxy #! read Contour HTTPProxy resources
        - --domain-filter=my-zone.example.org #! zone where services are deployed
        - --provider=aws
        - --policy=upsert-only #! prevent deleting any records, omit to enable full
synchronization
        - --aws-zone-type=public #! only look at public hosted zones (public, private, no
value for both)
        - --aws-prefer-cname
        - --registry=txt
        - --txt-owner-id=HOSTED_ZONE_ID #! Route53 hosted zone identifier for my-
zone.example.org
        - --txt-prefix=txt #! disambiguates TXT records from CNAME records
        env:
          - name: AWS_ACCESS_KEY_ID
            valueFrom:
              secretKeyRef:
                name: route53-credentials #! Kubernetes secret for route53 credentials
                key: aws_access_key_id
          - name: AWS_SECRET_ACCESS_KEY
```

```
            valueFrom:
              secretKeyRef:
                name: route53-credentials #! Kubernetes secret for route53 credentials
                key: aws_secret_access_key
```

Das folgende Beispiel kann für einen Azure DNS-Anbieter verwendet werden.

```
---
    namespace: service-discovery
    dns:
      pspNames: "vmware-system-restricted"
      deployment:
        args:
        - --provider=azure
        - --source=service
        - --source=ingress
        - --source=contour-httpproxy #! read Contour HTTPProxy resources
        - --domain-filter=my-zone.example.org #! zone where services are deployed
        - --azure-resource-group=my-resource-group #! Azure resource group
        volumeMounts:
        - name: azure-config-file
          mountPath: /etc/kubernetes
          readOnly: true
        #@overlay/replace
        volumes:
        - name: azure-config-file
          secret:
            secretName: azure-config-file
```

ExternalDNS-Konfiguration

In der Tabelle sind die verfügbaren Konfigurationsparameter für ExternalDNS aufgeführt und beschrieben. Weitere Informationen finden Sie auf der Site https://github.com/kubernetes-sigs/external-dns#running-externaldns.

Tabelle 11-3. Konfiguration des externen DNS-Pakets

Parameter	Beschreibung	Typ	Standard
externalDns.namespace	Namespace, in dem external-dns bereitgestellt wird	string	tanzu-system-service-discovery
externalDns.image.repository	Repository mit external-dns-Image	string	projects.registry.vmware.com/tkg
externalDns.image.name	Name von external-dns	string	external-dns
externalDns.image.tag	ExternalDNS-Image-Tag	string	v0.7.4_vmware.1
externalDns.image.pullPolicy	Pull-Richtlinie für das ExternalDNS-Image	string	IfNotPresent
externalDns.deployment.annotations	Anmerkungen zur Bereitstellung von external-dns	map<string,string>	{}

Tabelle 11-3. Konfiguration des externen DNS-Pakets (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
externalDns.deployment.args	Über die Befehlszeile an external-dns übergebene Argumente	list<string>	`[]` (Obligatorischer Parameter)
externalDns.deployment.env	An external-dns zu übergebende Umgebungsvariablen	list<string>	[]
externalDns.deployment.securityContext	Sicherheitskontext des external-dns-Containers	SecurityContext	{}
externalDns.deployment.volumeMounts	Volume-Mounts des external-dns-Containers	list<VolumeMount>	[]
externalDns.deployment.volumes	Volumes des external-dns-Pods	list<Volume>	[]

Beispiel für ConfigMap

Das folgende Beispiel für ConfigMap definiert eine Kerberos-Konfiguration, mit der ExternalDNS verbunden werden kann. Benutzerdefinierte Einträge enthalten den Domänen-/Bereichsnamen und die kdc/admin_server-Adressen.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: krb.conf
  namespace: tanzu-system-service-discovery
data:
  krb5.conf: |
    [logging]
    default = FILE:/var/log/krb5libs.log
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmind.log

    [libdefaults]
    dns_lookup_realm = false
    ticket_lifetime = 24h
    renew_lifetime = 7d
    forwardable = true
    rdns = false
    pkinit_anchors = /etc/pki/tls/certs/ca-bundle.crt
    default_ccache_name = KEYRING:persistent:%{uid}

    default_realm = CORP.ACME

    [realms]
    CORP.ACME = {
      kdc = controlcenter.corp.acme
      admin_server = controlcenter.corp.acme
    }
```

```
    [domain_realm]
    corp.acme = CORP.ACME
    .corp.acme = CORP.ACME
```

Referenz zum Fluent Bit-Paket

Dieses Thema enthält Referenzinformationen für das Fluent Bit-Paket.

Informationen zu Fluent Bit

Fluent Bit (https://fluentbit.io/) ist ein schneller, schlanker Protokollprozessor sowie eine schnelle Protokollweiterleitungsfunktion, mit der Sie Anwendungsdaten und -protokolle aus verschiedenen Quellen erfassen, vereinheitlichen und an mehrere Ziele senden können.

Sie können Fluent Bit als Protokollweiterleitung für Protokolle aus einem TKG-Cluster verwenden. Dabei muss ein Verwaltungsserver für die Protokollierung zum Speichern und Analysieren von Protokollen bereitgestellt sein. Unterstützt werden unter anderem die Protokollierungsserver Syslog, HTTP, Elastic Search, Kafka und Splunk.

Informationen zum Installieren des Fluent Bit-Pakets in einem TKG-Cluster finden Sie in den folgenden Themen:

- vSphere 8.x TKr: Installieren von Fluent Bit
- vSphere 7.x TKr: #unique_175

Fluent Bit-Komponenten

Das Fluent Bit-Paket installiert den in der Tabelle aufgeführten Container auf dem Cluster. Das Paket ruft den Container aus der öffentlichen Registrierung ab, die im Paket-Repository angegeben ist.

Container	Ressourcentyp	Replikate	Beschreibung
Fluent Bit	DaemonSet	6	Protokoll-Collector, Aggregator, Weiterleitungsfunktion

Fluent Bit-Datenwerte

Das folgende Beispiel für `fluent-bit-data-values.yaml` kann für einen Syslog-Server verwendet werden.

```
---
namespace: fluentbit-logging
tkg:
  instance_name: "<TKG_INSTANCE_NAME>"
  cluster_name: "<CLUSTER_NAME>"
fluentbit:
  output_plugin: "syslog"
  syslog:
```

```
    host: "<SYSLOG_HOST>"
    port: "<SYSLOG_PORT>"
    mode: "<SYSLOG_MODE>"
    format: "<SYSLOG_FORMAT>"
```

Das folgende Beispiel für `fluent-bit-data-values.yaml` kann für HTTP-Endpoints verwendet werden.

```
---
namespace: fluentbit-logging
tkg:
  instance_name: "<TKG_INSTANCE_NAME>"
  cluster_name: "<CLUSTER_NAME>"
fluentbit:
  output_plugin: "http"
  http:
    host: "<HTTP_HOST>"
    port: "<HTTP_PORT>"
    uri: "<URI>"
    header_key_value: "<HEADER_KEY_VALUE>"
    format: "json"
```

Das folgende Beispiel für `fluent-bit-data-values.yaml` kann für Elastic Search verwendet werden.

```
---
namespace: fluentbit-logging
tkg:
  instance_name: "<TKG_INSTANCE_NAME>"
  cluster_name: "<CLUSTER_NAME>"
fluentbit:
  output_plugin: "elasticsearch"
  elasticsearch:
    host: "<ELASTIC_SEARCH_HOST>"
    port: "<ELASTIC_SEARCH_PORT>"
```

Das folgende Beispiel für `fluent-bit-data-values.yaml` kann für Kafka verwendet werden.

```
---
namespace: fluentbit-logging
tkg:
  instance_name: "<TKG_INSTANCE_NAME>"
  cluster_name: "<CLUSTER_NAME>"
fluentbit:
  output_plugin: "kafka"
  kafka:
    broker_service_name: "<BROKER_SERVICE_NAME>"
    topic_name: "<TOPIC_NAME>"
```

Das folgende Beispiel für `fluent-bit-data-values.yaml` kann für Splunk verwendet werden.

```
---
namespace: fluentbit-logging
tkg:
  instance_name: "<TKG_INSTANCE_NAME>"
  cluster_name: "<CLUSTER_NAME>"
fluentbit:
  output_plugin: "splunk"
  splunk:
    host: "<SPLUNK_HOST>"
    port: "<SPLUNK_PORT>"
    token: "<SPLUNK_TOKEN>"
```

Fluent Bit-Konfiguration

Die Konfigurationswerte werden in `fluent-bit-data-values.yaml` festgelegt. In der Tabelle sind die verfügbaren Parameter aufgeführt und beschrieben.

Tabelle 11-4. Konfigurationen des Fluent Bit-Pakets

Parameter	Beschreibung	Typ	Standard
logging.namespace	Namespace, in dem Fluent Bit bereitgestellt wird	string	tanzu-system-logging
logging.service_account_name	Name des Fluent Bit-Dienstkontos	string	fluent-bit
logging.cluster_role_name	Name der Clusterrolle, die Fluent Bit „Get“-, „Watch“- und „List“-Berechtigungen gewährt	string	fluent-bit-read
logging.image.name	Name des Fluent Bit-Images	string	fluent-bit
logging.image.tag	Fluent Bit-Image-Tag. Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Version durchführen.	string	v1.6.9_vmware.1
logging.image.repository	Speicherort des Repositorys mit dem Fluent Bit-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg
logging.image.pullPolicy	Pull-Richtlinie des Fluent Bit-Images	string	IfNotPresent

Tabelle 11-4. Konfigurationen des Fluent Bit-Pakets (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
logging.update_strategy	Update-Strategie, die bei der DaemonSet-Aktualisierung verwendet werden soll	string	RollingUpdate
tkg.cluster_name	Name des Tanzu Kubernetes-Clusters	string	Null (Obligatorischer Parameter)
tkg.instance_name	Benutzerdefinierter Name der TKG-Instanz, der vom Supervisor-Cluster und allen Tanzu Kubernetes-Clustern in einer Bereitstellung gemeinsam genutzt wird. Sie können einen beliebigen Namen im Zusammenhang mit der Installation verwenden.	string	Null (Obligatorischer Parameter) **Hinweis** Dieses Feld ist obligatorisch, aber beliebig. Es handelt sich um einen Namen, der in den Protokollen angezeigt wird.
fluent_bit.log_level	Zu verwendende Protokollebene für Fluent Bit	string	Info
fluent_bit.output_plugin	Legen Sie das Backend fest, an das Fluent Bit die erfassten Informationen übermitteln soll	string	Null (Obligatorischer Parameter)
fluent_bit.elasticsearch.host	IP-Adresse oder Hostname der Elasticsearch-Zielinstanz	string	Null (obligatorischer Parameter, wenn output_plugin eine elastische Suche ist)
fluent_bit.elasticsearch.port	TCP-Port der Elasticsearch-Zielinstanz	integer	Null (obligatorischer Parameter, wenn output_plugin eine elastische Suche ist)
fluent_bit.elasticsearch.buffer_size	Geben Sie die Puffergröße an, die zum Lesen der Antwort des Elasticsearch-Diensts verwendet wird. Wird auf „Unbegrenzt“ festgelegt, wenn „False“	string	False
fluent_bit.elasticsearch.tls	Geben Sie die Standardeinstellung für TLS für Elasticsearch an	string	Aus
fluent_bit.kafka.broker_service_name	Eine aus mehreren Listen mit Kafka-Brokern, z. B.: 192.168.1.3:9092	string	Null (obligatorischer Parameter, wenn output_plugin Kafka ist)

Tabelle 11-4. Konfigurationen des Fluent Bit-Pakets (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
fluent_bit.kafka.topic_name	Ein einzelner Eintrag oder eine Liste von Themen, die durch (,) getrennt sind, die Fluent Bit zum Senden von Nachrichten an Kafka vrwendet	string	Null (obligatorischer Parameter, wenn output_plugin Kafka ist)
fluent_bit.splunk.host	IP-Adresse oder Hostname des Splunk-Zielservers	string	Null (obligatorischer Parameter, wenn output_plugin Splunk ist)
fluent_bit.splunk.port	TCP-Port des Splunk-Zielservers	integer	Null (obligatorischer Parameter, wenn output_plugin Splunk ist)
fluent_bit.splunk.token	Angabe des Authentifizierungstokens für die HTTP Event Collector-Schnittstelle	string	Null (obligatorischer Parameter, wenn output_plugin Splunk ist)
fluent_bit.http.host	IP-Adresse oder Hostname des HTTP-Zielservers	string	Null (obligatorischer Parameter, wenn output_plugin http ist)
fluent_bit.http.port	TCP-Port des HTTP-Zielservers	integer	Null (obligatorischer Parameter, wenn output_plugin http ist)
fluent_bit.http.mode	Angabe einer HTTP-URl für den Zielwebserver	string	Null (obligatorischer Parameter, wenn output_plugin http ist)
fluent_bit.http.header_key_value	Schlüssel/Wert-Paar des HTTP-Headers. Es können mehrere Kopfzeilen festgelegt werden	string	Null (obligatorischer Parameter, wenn output_plugin http ist)
fluent_bit.http.format	Geben Sie das Datenformat an, das im Textkörper der HTTP-Anforderung verwendet werden soll	string	Null (obligatorischer Parameter, wenn output_plugin http ist)
fluent_bit.syslog.host	Domäne oder IP-Adresse des Remote-Syslog-Servers	string	Null (obligatorischer Parameter, wenn output_plugin Syslog ist)
fluent_bit.syslog.port	TCP- oder UDP-Port des Remote-Syslog-Servers	integer	Null (obligatorischer Parameter, wenn output_plugin Syslog ist)
fluent_bit.syslog.mode	Angabe des Transporttyps von TCP, UDP und TLS	string	Null (obligatorischer Parameter, wenn output_plugin Syslog ist)

Tabelle 11-4. Konfigurationen des Fluent Bit-Pakets (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
fluent_bit.syslog.format	Geben Sie das Datenformat an, das im Textkörper der HTTP-Anforderung verwendet werden soll	string	Null (obligatorischer Parameter, wenn output_plugin Syslog ist)
host_path.volume_1	Verzeichnispfad vom Dateisystem des Hostknotens in den Pod, für Volume 1	string	/var/log
host_path.volume_2	Verzeichnispfad vom Dateisystem des Hostknotens in den Pod, für Volume 2	string	/var/lib/docker/containers
host_path.volume_3	Verzeichnispfad vom Dateisystem des Hostknotens in den Pod, für Volume 3	string	/run/log
systemd.path	Pfad zum Systemd-Journalverzeichnis	string	/var/log/journal

Prometheus-Paketreferenz

Dieses Thema enthält Referenzinformationen für das Prometheus-Paket.

Informationen zu Prometheus und Alertmanager

Prometheus (https://prometheus.io/) ist ein System- und Dienstüberwachungssystem. Prometheus erfasst Metriken von konfigurierten Zielen in bestimmten Intervallen, wertet Regelausdrücke aus und zeigt die Ergebnisse an. Mithilfe von Alertmanager werden Warnungen ausgelöst, wenn festgestellt wird, dass eine Bedingung wahr ist.

So installieren Sie das Prometheus-Paket:

-
-

Prometheus-Komponenten

Das Prometheus-Paket installiert die in der Tabelle aufgeführten Container auf einem TKG-Cluster. Das Paket ruft die Container aus der öffentlichen VMware-Registrierung ab, die im Paket-Repository angegeben ist.

Container	Ressourcentyp	Replikate	Beschreibung
`prometheus-alertmanager`	Bereitstellung	1	Verarbeitet Warnungen, die von Clientanwendungen wie dem Prometheus-Server gesendet werden.
`prometheus-cadvisor`	DaemonSet	5	Analysiert und zeigt Ressourcennutzungs- und Leistungsdaten aus ausgeführten Containern an
`prometheus-kube-state-metrics`	Bereitstellung	1	Überwacht Knotenstatus und -kapazität, Replikat-Set-Konformität, Pod-, Auftrags- und Cronjob-Status, Ressourcenanforderungen und Grenzwerte.
`prometheus-node-exporter`	DaemonSet	5	Exporter für Hardware- und BS-Metriken, die von Kerneln verfügbar gemacht werden.
`prometheus-pushgateway`	Bereitstellung	1	Dienst, der es Ihnen ermöglicht, Metriken von Aufträgen zu übertragen, die nicht entfernt werden können.
`prometheus-server`	Bereitstellung	1	Bietet grundlegende Funktionen, einschließlich Scraping, Regelverarbeitung und Warnungen.

Prometheus-Datenwerte

Nachfolgend finden Sie ein beispielhafte `prometheus-data-values.yaml`-Datei.

Beachten Sie Folgendes:

- Ingress ist aktiviert (ingress: enabled: true).
- Ingress ist für URLs konfiguriert, die auf /alertmanager/ (alertmanager_prefix:) und / (prometheus_prefix:) enden.
- Der FQDN für Prometheus lautet `prometheus.system.tanzu` (virtual_host_fqdn:).
- Stellen Sie Ihr eigenes benutzerdefiniertes Zertifikat im Ingress-Abschnitt bereit (tls.crt, tls.key, ca.crt).
- Die PVC für alertmanager beläuft sich auf 2 GiB. Geben Sie den `storageClassName` für die Standardspeicherrichtlinie an.
- Die PVC für Prometheus beläuft sich auf 20 GiB. Geben Sie den `storageClassName` für die vSphere-Speicherrichtlinie an.

```
namespace: prometheus-monitoring
alertmanager:
  config:
    alertmanager_yml: |
      global: {}
      receivers:
      - name: default-receiver
      templates:
      - '/etc/alertmanager/templates/*.tmpl'
      route:
```

```
        group_interval: 5m
        group_wait: 10s
        receiver: default-receiver
        repeat_interval: 3h
  deployment:
    replicas: 1
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 2Gi
    storageClassName: default
  service:
    port: 80
    targetPort: 9093
    type: ClusterIP
ingress:
  alertmanager_prefix: /alertmanager/
  alertmanagerServicePort: 80
  enabled: true
  prometheus_prefix: /
  prometheusServicePort: 80
  tlsCertificate:
    ca.crt: |
      -----BEGIN CERTIFICATE-----
      MIIFczCCA1ugAwIBAgIQTYJITQ3SZ4BBS9UzXfJIuTANBgkqhkiG9w0BAQsFADBM
      ...
      w0oGuTTBfxSMKs767N3G1q5tz0mwFpIqIQtXUSmaJ+9p7IkpWcThLnyYYo1IpWm/
      ZHtjzZMQVA==
      -----END CERTIFICATE-----
    tls.crt: |
      -----BEGIN CERTIFICATE-----
      MIIHxTCCBa2gAwIBAgITIgAAAAQnSpH7QfxTKAAAAAAABDANBgkqhkiG9w0BAQsF
      ...
      YYsIjp7/f+Pk1DjzWx8JIAbzItKLucDreAmmDXqk+DrBP9LYqtmjB0n7nSErgK8G
      sA3kGCJdOkI0kgF10gsinaouG2jVlwNOsw==
      -----END CERTIFICATE-----
    tls.key: |
      -----BEGIN PRIVATE KEY-----
      MIIJRAIBADANBgkqhkiG9w0BAQEFAASCCS4wggkqAgEAAoICAQDOGHT8I12KyQGS
      ...
      l1NzswracGQIzo03zk/X3Z6P2YOea4BkZ0Iwh34wOHJnTkfEeSx6y+oSFMcFRthT
      yfFCZUk/sVCc/C1a4VigczXftUGiRrTR
      -----END PRIVATE KEY-----
  virtual_host_fqdn: prometheus.system.tanzu
kube_state_metrics:
  deployment:
    replicas: 1
  service:
    port: 80
    targetPort: 8080
    telemetryPort: 81
    telemetryTargetPort: 8081
```

```
    type: ClusterIP
node_exporter:
  daemonset:
    hostNetwork: false
    updatestrategy: RollingUpdate
  service:
    port: 9100
    targetPort: 9100
    type: ClusterIP
prometheus:
  pspNames: "vmware-system-restricted"
  config:
    alerting_rules_yml: |
      {}
    alerts_yml: |
      {}
    prometheus_yml: |
      global:
        evaluation_interval: 1m
        scrape_interval: 1m
        scrape_timeout: 10s
      rule_files:
      - /etc/config/alerting_rules.yml
      - /etc/config/recording_rules.yml
      - /etc/config/alerts
      - /etc/config/rules
      scrape_configs:
      - job_name: 'prometheus'
        scrape_interval: 5s
        static_configs:
        - targets: ['localhost:9090']
      - job_name: 'kube-state-metrics'
        static_configs:
        - targets: ['prometheus-kube-state-metrics.prometheus.svc.cluster.local:8080']

      - job_name: 'node-exporter'
        static_configs:
        - targets: ['prometheus-node-exporter.prometheus.svc.cluster.local:9100']

      - job_name: 'kubernetes-pods'
        kubernetes_sd_configs:
        - role: pod
        relabel_configs:
        - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
          action: keep
          regex: true
        - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
          action: replace
          target_label: __metrics_path__
          regex: (.+)
        - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
          action: replace
          regex: ([^:]+)(?::\d+)?;(\d+)
          replacement: $1:$2
          target_label: __address__
```

```
    - action: labelmap
      regex: __meta_kubernetes_pod_label_(.+)
    - source_labels: [__meta_kubernetes_namespace]
      action: replace
      target_label: kubernetes_namespace
    - source_labels: [__meta_kubernetes_pod_name]
      action: replace
      target_label: kubernetes_pod_name
  - job_name: kubernetes-nodes-cadvisor
    kubernetes_sd_configs:
    - role: node
    relabel_configs:
    - action: labelmap
      regex: __meta_kubernetes_node_label_(.+)
    - replacement: kubernetes.default.svc:443
      target_label: __address__
    - regex: (.+)
      replacement: /api/v1/nodes/$1/proxy/metrics/cadvisor
      source_labels:
      - __meta_kubernetes_node_name
      target_label: __metrics_path__
    scheme: https
    tls_config:
      ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      insecure_skip_verify: true
    bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
  - job_name: kubernetes-apiservers
    kubernetes_sd_configs:
    - role: endpoints
    relabel_configs:
    - action: keep
      regex: default;kubernetes;https
      source_labels:
      - __meta_kubernetes_namespace
      - __meta_kubernetes_service_name
      - __meta_kubernetes_endpoint_port_name
    scheme: https
    tls_config:
      ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      insecure_skip_verify: true
    bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
  alerting:
    alertmanagers:
    - scheme: http
      static_configs:
      - targets:
        - alertmanager.prometheus.svc:80
    - kubernetes_sd_configs:
        - role: pod
      relabel_configs:
      - source_labels: [__meta_kubernetes_namespace]
        regex: default
        action: keep
      - source_labels: [__meta_kubernetes_pod_label_app]
        regex: prometheus
```

```
            action: keep
          - source_labels: [__meta_kubernetes_pod_label_component]
            regex: alertmanager
            action: keep
          - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_probe]
            regex: .*
            action: keep
          - source_labels: [__meta_kubernetes_pod_container_port_number]
            regex:
            action: drop
    recording_rules_yml: |
      groups:
        - name: kube-apiserver.rules
          interval: 3m
          rules:
          - expr: |2
              (
                (
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[1d]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[1d]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[1d]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[1d]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[1d]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[1d]))
            labels:
              verb: read
            record: apiserver_request:burnrate1d
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[1h]))
                  -
                  (
```

```
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[1h]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[1h]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[1h]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[1h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[1h]))
            labels:
              verb: read
            record: apiserver_request:burnrate1h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[2h]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[2h]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[2h]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[2h]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[2h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[2h]))
```

```
            labels:
              verb: read
            record: apiserver_request:burnrate2h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[30m]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[30m]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[30m]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[30m]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[30m]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[30m]))
            labels:
              verb: read
            record: apiserver_request:burnrate30m
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[3d]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[3d]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[3d]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
```

```
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[3d]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[3d]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[3d]))
            labels:
              verb: read
            record: apiserver_request:burnrate3d
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[5m]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[5m]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[5m]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[5m]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[5m]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[5m]))
            labels:
              verb: read
            record: apiserver_request:burnrate5m
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[6h]))
                  -
                  (
```

```
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[6h]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[6h]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[6h]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|
GET",code=~"5.."}[6h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST|GET"}
[6h]))
            labels:
              verb: read
            record: apiserver_request:burnrate6h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[1d]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[1d]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[1d]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[1d]))
            labels:
              verb: write
            record: apiserver_request:burnrate1d
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[1h]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[1h]))
                )
```

```
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[1h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[1h]))
            labels:
              verb: write
            record: apiserver_request:burnrate1h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[2h]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[2h]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[2h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[2h]))
            labels:
              verb: write
            record: apiserver_request:burnrate2h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[30m]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[30m]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[30m]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[30m]))
            labels:
              verb: write
            record: apiserver_request:burnrate30m
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
```

```
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[3d]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[3d]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[3d]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[3d]))
            labels:
              verb: write
            record: apiserver_request:burnrate3d
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[5m]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[5m]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[5m]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[5m]))
            labels:
              verb: write
            record: apiserver_request:burnrate5m
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[6h]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[6h]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE",code=~"5.."}[6h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST|PUT|
PATCH|DELETE"}[6h]))
            labels:
              verb: write
            record: apiserver_request:burnrate6h
```

```
          - expr: |
              sum by (code,resource) (rate(apiserver_request_total{job="kubernetes-
apiservers",verb=~"LIST|GET"}[5m]))
            labels:
              verb: read
            record: code_resource:apiserver_request_total:rate5m
          - expr: |
              sum by (code,resource) (rate(apiserver_request_total{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[5m]))
            labels:
              verb: write
            record: code_resource:apiserver_request_total:rate5m
          - expr: |
              histogram_quantile(0.99, sum by
(le, resource) (rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET"}[5m]))) > 0
            labels:
              quantile: "0.99"
              verb: read
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile
          - expr: |
              histogram_quantile(0.99, sum by
(le, resource) (rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"POST|PUT|PATCH|DELETE"}[5m]))) > 0
            labels:
              quantile: "0.99"
              verb: write
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile
          - expr: |2
              sum(rate(apiserver_request_duration_seconds_sum{subresource!="log",verb!~"LIST|
WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}[5m])) without(instance, pod)
              /
              sum(rate(apiserver_request_duration_seconds_count{subresource!="log",verb!
~"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}[5m])) without(instance, pod)
            record: cluster:apiserver_request_duration_seconds:mean5m
          - expr: |
              histogram_quantile(0.99,
sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-apiservers",subresource!
="log",verb!~"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}[5m])) without(instance,
pod))
            labels:
              quantile: "0.99"
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile
          - expr: |

histogram_quantile(0.9, sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",subresource!="log",verb!~"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}
[5m])) without(instance, pod))
            labels:
              quantile: "0.9"
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile
          - expr: |

histogram_quantile(0.5, sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",subresource!="log",verb!~"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}
```

```
[5m])) without(instance, pod))
            labels:
              quantile: "0.5"
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile
        - interval: 3m
          name: kube-apiserver-availability.rules
          rules:
          - expr: |2
              1 - (
                (
                  # write too slow
                  sum(increase(apiserver_request_duration_seconds_count{verb=~"POST|PUT|PATCH|
DELETE"}[30d]))
                  -
                  sum(increase(apiserver_request_duration_seconds_bucket{verb=~"POST|PUT|
PATCH|DELETE",le="1"}[30d]))
                ) +
                (
                  # read too slow
                  sum(increase(apiserver_request_duration_seconds_count{verb=~"LIST|GET"}
[30d]))
                  -
                  (
                    (
                      sum(increase(apiserver_request_duration_seconds_bucket{verb=~"LIST|
GET",scope=~"resource|",le="0.1"}[30d]))
                      or
                      vector(0)
                    )
                    +
                    sum(increase(apiserver_request_duration_seconds_bucket{verb=~"LIST|
GET",scope="namespace",le="0.5"}[30d]))
                    +
                    sum(increase(apiserver_request_duration_seconds_bucket{verb=~"LIST|
GET",scope="cluster",le="5"}[30d]))
                  )
                ) +
                # errors
                sum(code:apiserver_request_total:increase30d{code=~"5.."} or vector(0))
              )
              /
              sum(code:apiserver_request_total:increase30d)
            labels:
              verb: all
            record: apiserver_request:availability30d
          - expr: |2
              1 - (
                sum(increase(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[30d]))
                -
                (
                  # too slow
                  (
                    sum(increase(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[30d]))
```

```
                    or
                    vector(0)
                  )
                  +
                  sum(increase(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[30d]))
                  +
                  sum(increase(apiserver_request_duration_seconds_bucket{job="kubernetes-
apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[30d]))
                )
                +
                # errors
                sum(code:apiserver_request_total:increase30d{verb="read",code=~"5.."} or
vector(0))
              )
              /
              sum(code:apiserver_request_total:increase30d{verb="read"})
            labels:
              verb: read
            record: apiserver_request:availability30d
          - expr: |2
              1 - (
                (
                  # too slow
                  sum(increase(apiserver_request_duration_seconds_count{verb=~"POST|PUT|PATCH|
DELETE"}[30d]))
                  -
                  sum(increase(apiserver_request_duration_seconds_bucket{verb=~"POST|PUT|
PATCH|DELETE",le="1"}[30d]))
                )
                +
                # errors
                sum(code:apiserver_request_total:increase30d{verb="write",code=~"5.."} or
vector(0))
              )
              /
              sum(code:apiserver_request_total:increase30d{verb="write"})
            labels:
              verb: write
            record: apiserver_request:availability30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="LIST",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="GET",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="POST",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PUT",code=~"2.."}[30d]))
```

```
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PATCH",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="DELETE",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="LIST",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="GET",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="POST",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PUT",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PATCH",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="DELETE",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="LIST",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="GET",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="POST",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PUT",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PATCH",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
```

```
apiservers",verb="DELETE",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="LIST",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="GET",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="POST",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PUT",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="PATCH",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-
apiservers",verb="DELETE",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code) (code_verb:apiserver_request_total:increase30d{verb=~"LIST|GET"})
            labels:
              verb: read
            record: code:apiserver_request_total:increase30d
          - expr: |
              sum by (code) (code_verb:apiserver_request_total:increase30d{verb=~"POST|
PUT|PATCH|DELETE"})
            labels:
              verb: write
            record: code:apiserver_request_total:increase30d
    rules_yml: |
      {}
  deployment:
    configmapReload:
      containers:
        args:
          - --volume-dir=/etc/config
          - --webhook-url=http://127.0.0.1:9090/-/reload
    containers:
      args:
        - --storage.tsdb.retention.time=42d
        - --config.file=/etc/config/prometheus.yml
        - --storage.tsdb.path=/data
        - --web.console.libraries=/etc/prometheus/console_libraries
        - --web.console.templates=/etc/prometheus/consoles
        - --web.enable-lifecycle
    replicas: 1
    rollingUpdate:
```

```
      maxSurge: 25%
      maxUnavailable: 25%
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 20Gi
    storageClassName: default
  service:
    port: 80
    targetPort: 9090
    type: ClusterIP
pushgateway:
  deployment:
    replicas: 1
  service:
    port: 9091
    targetPort: 9091
    type: ClusterIP
```

Prometheus-Konfiguration

Die Prometheus-Konfiguration wird in der `prometheus-data-values.yaml`-Datei festgelegt. In der Tabelle sind die verfügbaren Parameter aufgeführt und beschrieben.

Tabelle 11-5. Prometheus-Konfigurationsparameter

Parameter	Beschreibung	Typ	Standard
monitoring.namespace	Namespace, in dem Prometheus bereitgestellt wird	string	tanzu-system-monitoring
monitoring.create_namespace	Das Flag gibt an, ob der durch monitoring.namespace angegebene Namespace erstellt werden soll.	Boolean	false
monitoring.prometheus_server.config.prometheus_yaml	Konfigurationsdetails der Kubernetes-Clusterüberwachung, die an Prometheus übergeben werden sollen	yaml-Datei	prometheus.yaml
monitoring.prometheus_server.config.alerting_rules_yaml	Detaillierte Warnungsregeln, die in Prometheus definiert sind	yaml-Datei	alerting_rules.yaml
monitoring.prometheus_server.config.recording_rules_yaml	Detaillierte Aufzeichnungsregeln, die in Prometheus definiert sind	yaml-Datei	recording_rules.yaml
monitoring.prometheus_server.service.type	Diensttyp, um Prometheus verfügbar zu machen. Unterstützte Werte: ClusterIP	string	ClusterIP

Tabelle 11-5. Prometheus-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
monitoring.prometheus_server.enable_alerts.kubernetes_api	Aktivieren der SLO-Warnung für die Kubernetes-API in Prometheus	Boolean	true
monitoring.prometheus_server.sc.aws_type	AWS-Typ, der für StorageClass auf AWS definiert ist	string	gp2
monitoring.prometheus_server.sc.aws_fsType	AWS-Dateisystemtyp, der für StorageClass auf AWS definiert ist	string	ext4
monitoring.prometheus_server.sc.allowVolumeExpansion	Legt fest, ob die Volume-Erweiterung für StorageClass auf AWS zulässig ist	Boolean	true
monitoring.prometheus_server.pvc.annotations	Anmerkungen zur Speicherklasse	Zuordnung	{}
monitoring.prometheus_server.pvc.storage_class	Für die Beanspruchung eines persistenten Volumes zu verwendende Speicherklasse. Standardmäßig ist diese null und es wird der Standard-Provisioner verwendet	string	null
monitoring.prometheus_server.pvc.accessMode	Definieren Sie den Zugriffsmodus für die Beanspruchung eines dauerhaften Volumes. Unterstützte Werte: ReadWriteOnce, ReadOnlyMany, ReadWriteMany	string	ReadWriteOnce
monitoring.prometheus_server.pvc.storage	Definition der Speichergröße für die Beanspruchung eines persistenten Volumes	string	8Gi
monitoring.prometheus_server.deployment.replicas	Anzahl an Prometheus-Replikas	integer	1

Tabelle 11-5. Prometheus-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
monitoring.prometheus_server.image.repository	Speicherort des Repositorys mit dem Prometheus-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg/prometheus
monitoring.prometheus_server.image.name	Name des Prometheus-Images	string	Prometheus
monitoring.prometheus_server.image.tag	Prometheus-Image-Tag. Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Version durchführen.	string	v2.17.1_vmware.1
monitoring.prometheus_server.image.pullPolicy	Prometheus-Image-Pull-Richtlinie	string	IfNotPresent
monitoring.alertmanager.config.slack_demo	Slack-Benachrichtigungskonfiguration für Alertmanager	string	`slack_demo:   name: slack_demo   slack_configs:   - api_url: https:// hooks.slack.com   channel: '#alertmanager- test'`
monitoring.alertmanager.config.email_receiver	E-Mail-Benachrichtigungskonfiguration für Alertmanager	string	`email_receiver:   name: email- receiver   email_configs:   - to: demo@tanzu.com   send_resolved: false   from: from- email@tanzu.com   smarthost: smtp.eample.com:25   require_tls: false`
monitoring.alertmanager.service.type	Diensttyp, um Alertmanager verfügbar zu machen. Unterstützte Werte: ClusterIP	string	ClusterIP

Tabelle 11-5. Prometheus-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
monitoring.alertmanager.image.repository	Speicherort des Repositorys mit dem Alertmanager-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg/prometheus
monitoring.alertmanager.image.name	Name des Alertmanager-Images	string	alertmanager
monitoring.alertmanager.image.tag	Alertmanager-Image-Tag. Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Version durchführen.	string	v0.20.0_vmware.1
monitoring.alertmanager.image.pullPolicy	Alertmanager-Image-Pull-Richtlinie	string	IfNotPresent
monitoring.alertmanager.pvc.annotations	Anmerkungen zur Speicherklasse	Zuordnung	{}
monitoring.alertmanager.pvc.storage_class	Für die Beanspruchung eines persistenten Volumes zu verwendende Speicherklasse. Standardmäßig ist diese null und es wird der Standard-Provisioner verwendet.	string	null
monitoring.alertmanager.pvc.accessMode	Definieren Sie den Zugriffsmodus für die Beanspruchung eines dauerhaften Volumes. Unterstützte Werte: ReadWriteOnce, ReadOnlyMany, ReadWriteMany	string	ReadWriteOnce
monitoring.alertmanager.pvc.storage	Definition der Speichergröße für die Beanspruchung eines persistenten Volumes	string	2Gi
monitoring.alertmanager.deployment.replicas	Anzahl an alertmanager-Replikas	integer	1

Tabelle 11-5. Prometheus-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
monitoring.kube_state_metrics.image.repository	Repository mit dem kube-state-metircs-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg/prometheus
monitoring.kube_state_metrics.image.name	Name des kube-state-metircs-Images	string	kube-state-metrics
monitoring.kube_state_metrics.image.tag	kube-state-metrics-Image-Tag. Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Version durchführen.	string	v1.9.5_vmware.1
monitoring.kube_state_metrics.image.pullPolicy	Kube-state-metrics-Image-Pull-Richtlinie	string	IfNotPresent
monitoring.kube_state_metrics.deployment.replicas	Anzahl an kube-state-metrics-Replikas	integer	1
monitoring.node_exporter.image.repository	Repository mit dem node-exporter-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg/prometheus
monitoring.node_exporter.image.name	Name des node-exporter-Images	string	node-exporter
monitoring.node_exporter.image.tag	node-exporter-Image-Tag. Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Version durchführen.	string	v0.18.1_vmware.1
monitoring.node_exporter.image.pullPolicy	node-exporter-Image-Pull-Richtlinie	string	IfNotPresent
monitoring.node_exporter.hostNetwork	Wenn dieser Wert auf `hostNetwork: true` festgelegt ist, kann der Pod den Netzwerk-Namespace und die Netzwerkressourcen des Knotens verwenden.	Boolean	false

Tabelle 11-5. Prometheus-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
monitoring.node_exporter.deployment.replicas	Anzahl an node-exporter-Replikas	integer	1
monitoring.pushgateway.image.repository	Repository mit dem pushgateway-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg/prometheus
monitoring.pushgateway.image.name	Name des pushgateway-Images	string	pushgateway
monitoring.pushgateway.image.tag	pushgateway-Image-Tag. Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Version durchführen.	string	v1.2.0_vmware.1
monitoring.pushgateway.image.pullPolicy	Pushgateway-Image-Pull-Richtlinie	string	IfNotPresent
monitoring.pushgateway.deployment.replicas	Anzahl der Pushgateway-Replikate	integer	1
monitoring.cadvisor.image.repository	Repository mit dem cadvisor-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg/prometheus
monitoring.cadvisor.image.name	Name des Cadvisor-Images	string	cadvisor
monitoring.cadvisor.image.tag	cadvisor-Image-Tag. Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Version durchführen.	string	v0.36.0_vmware.1
monitoring.cadvisor.image.pullPolicy	Cadvisor-Image-Pull-Richtlinie	string	IfNotPresent
monitoring.cadvisor.deployment.replicas	Anzahl an Cadvisor-Replikas	integer	1

Tabelle 11-5. Prometheus-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
monitoring.ingress.enabled	Ingress für Prometheus und Alertmanager aktivieren/deaktivieren	Boolean	false Zur Verwendung von Ingress setzen Sie dieses Feld auf `true` und stellen Sie Contour bereit. Aktualisieren Sie für den Zugriff auf Prometheus Ihre lokalen `/etc/hosts` mit einem Eintrag, der `prometheus.system.tanzu` einer IP-Adresse des Worker-Knotens zuordnet.
monitoring.ingress.virtual_host_fqdn	Hostname für den Zugriff auf Prometheus und Alertmanager	string	prometheus.system.tanzu
monitoring.ingress.prometheus_prefix	Pfadpräfix für Prometheus	string	/
monitoring.ingress.alertmanager_prefix	Pfadpräfix für Alertmanager	string	/alertmanager/
monitoring.ingress.tlsCertificate.tls.crt	Optionales Zertifikat für Ingress, wenn Sie Ihr eigenes TLS-Zertifikat verwenden möchten. Standardmäßig wird ein selbstsigniertes Zertifikat generiert	string	Generiertes Zertifikat
monitoring.ingress.tlsCertificate.tls.key	Optionaler privater Zertifikatsschlüssel für Ingress, wenn Sie Ihr eigenes TLS-Zertifikat verwenden möchten.	string	Generierter Zertifikatsschlüssel

Tabelle 11-6. Konfigurierbare Felder für Prometheus_Server-Configmap

Parameter	Beschreibung	Typ	Standard
evaluation_interval	Häufigkeit, mit der Regeln ausgewertet werden	Dauer	1 m
scrape_interval	Häufigkeit des Scrapens von Zielen	Dauer	1 m
scrape_timeout	Dauer bis zum Timeout einer Scrape-Anforderung	Dauer	10 s

Tabelle 11-6. Konfigurierbare Felder für Prometheus_Server-Configmap (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
rule_files	Regeldateien geben eine Liste von Globs an. Regeln und Warnungen werden aus allen übereinstimmenden Dateien gelesen	yaml-Datei	
scrape_configs	Eine Liste mit Scraper-Konfigurationen.	Liste	
job_name	Der Jobname, der scraped-Metriken standardmäßig zugewiesen ist	string	
kubernetes_sd_configs	Liste der Kubernetes-Diensterkennungskonfigurationen.	Liste	
relabel_configs	Liste der Konfigurationen für die erneute Beschriftung des Ziels.	Liste	
Aktion	Durchzuführende Aktion basierend auf dem regex-Abgleich.	string	
regex	Regulärer Ausdruck, mit dem der extrahierte Wert übereinstimmen soll.	string	
source_labels	Die Quellbezeichnungen wählen Werte aus vorhandenen Bezeichnungen aus.	string	
scheme	Konfiguriert das für Anforderungen verwendete Protokollschema.	string	
tls_config	Konfiguriert die TLS-Einstellungen der Scraper-Anforderung.	string	
ca_file	CA-Zertifikat zur Validierung des API-Serverzertifikats.	Dateiname	
insecure_skip_verify	Deaktivieren Sie die Validierung des Serverzertifikats.	Boolean	
bearer_token_file	Optionale Authentifizierungsinformationen zur Bearer-Token-Datei.	Dateiname	

Tabelle 11-6. Konfigurierbare Felder für Prometheus_Server-Configmap (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
replacement	Ersatzwert, mit dem ein Regex ersetzt wird, wenn der reguläre Ausdruck stimmt.	string	
target_label	Bezeichnung, in die der resultierende Wert in einer Ersatzaktion geschrieben wird.	string	

Tabelle 11-7. Konfigurierbare Felder für Alertmanager-Configmap

Parameter	Beschreibung	Typ	Standard
resolve_timeout	ResolveTimeout ist der Standardwert, der von alertmanager verwendet wird, wenn EndsAt nicht in der Warnung enthalten ist	Dauer	5 m
smtp_smarthost	Der SMTP-Host, über den E-Mails gesendet werden.	Dauer	1 m
slack_api_url	Die Slack-Webhook-URL.	string	global.slack_api_url
pagerduty_url	Die Pagerduty-URL, an die API-Anforderungen gesendet werden.	string	global.pagerduty_url
Vorlagen	Dateien, aus denen benutzerdefinierte Benachrichtigungsvorlagen definitionen gelesen werden	Dateipfad	
group_by	Warnungen, nach Bezeichnung gruppiert	string	
group_interval	Festgelegte Wartezeit, bevor eine Benachrichtigung zu neuen Warnungen gesendet wird, die zu einer Gruppe hinzugefügt werden	Dauer	5 m
group_wait	Wie lange zunächst auf das Senden einer Benachrichtigung für eine Gruppe von Warnungen gewartet werden soll	Dauer	30 s

Tabelle 11-7. Konfigurierbare Felder für Alertmanager-Configmap (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
repeat_interval	Wie lange gewartet werden soll, bis eine Benachrichtigung erneut gesendet wird, wenn sie bereits erfolgreich für eine Warnung gesendet wurde	Dauer	4 h
Empfänger	Eine Liste der Benachrichtigungsempfänger.	Liste	
Schweregrad	Schweregrad des Vorfalls	string	
Kanal	Der Kanal oder Benutzer, an den Benachrichtigungen gesendet werden sollen.	string	
html	Der HTML-Textkörper der E-Mail-Benachrichtigung.	string	
Text	Der Textkörper der E-Mail-Benachrichtigung.	string	
send_resolved	Ob behobene Warnungen gemeldet werden sollen oder nicht.	Dateiname	
email_configs	Konfigurationen für E-Mail-Integration	Boolean	

Anmerkungen auf Pods ermöglichen die Feinsteuerung des Scraping-Vorgangs. Diese Anmerkungen müssen Teil der Pod-Metadaten sein. Sie haben keine Auswirkung, wenn sie auf andere Objekte wie z. B. Dienste oder DaemonSets festgelegt sind.

Tabelle 11-8. Prometheus-Pod-Anmerkungen

Pod-Anmerkung	Beschreibung
`prometheus.io/scrape`	Mit der Standardkonfiguration werden alle Pods entfernt. Wenn sie auf „false“ festgelegt ist, schließt diese Anmerkung den Pod vom Scraping-Vorgang aus.
`prometheus.io/path`	Wenn der Metrikpfad nicht "/metrics" ist, definieren Sie ihn mit dieser Anmerkung.
`prometheus.io/port`	Scrapen Sie den Pod auf dem angegebenen Port anstatt auf den deklarierten Ports des Pods (der Standard ist ein Ziel ohne Port, wenn keiner angegeben ist).

Das unten aufgeführte DaemonSet-Manifest weist Prometheus an, alle seine Pods auf Port 9102 zu scrapen.

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use extensions/v1beta1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: weave
  labels:
    app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/port: '9102'
    spec:
      containers:
      - name: fluentd-elasticsearch
        image: gcr.io/google-containers/fluentd-elasticsearch:1.20
```

Referenz zum Grafana-Paket

Dieses Thema enthält Referenzinformationen für die Konfiguration von Grafana.

Informationen zu Grafana

Grafana (https://grafana.com/) ist eine Open Source-Visualisierungs- und Analysesoftware. Mit Grafana können Sie Metriken unabhängig davon, wo sie gespeichert sind, abfragen, visualisieren, erkunden und entsprechende Warnungen ausgeben. Grafana bietet Tools zum Erstellen von Diagrammen und Visualisierungen aus Anwendungsdaten.

Informationen zum Installieren des Grafana-Pakets auf einem TKG-Cluster finden Sie in folgenden Themen:

- TKr für vSphere 8.x: Installieren von Grafana
- TKr für vSphere 7.x: #unique_177

Komponenten des Grafana-Pakets

Das Grafana-Paket installiert den in der Tabelle aufgeführten Container auf dem Cluster. Das Grafana-Paket ruft den Container aus der öffentlichen Registrierung ab, die im Paket-Repository angegeben ist.

Container	Ressourcentyp	Replikate	Beschreibung
Grafana	Bereitstellung	2	Datenvisualisierung

Grafana-Datenwerte

Nachfolgend finden Sie eine beispielhafte `grafana-data-values.yaml`-Datei mit den folgenden Anpassungen:

- Ingress ist aktiviert (Ingress: aktiviert: true)
- Ingress ist für URLs konfiguriert, die auf / enden (Präfix:)
- Der FQDN für Grafana lautet grafana.system.tanzu (virtual_host_fqdn:)
- Die PVC für Grafana beläuft sich auf 2 GB und wird unter der standardmäßigen StorageClass in vSphere erstellt.
- Das Administratorkennwort (base64-codiert) für die Grafana-Benutzeroberfläche (grafana: geheim: admin_password:).

```
namespace: grafana-dashboard
grafana:
  deployment:
    replicas: 1
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 2Gi
    storageClassName: default
  secret:
    admin_password: admin
    admin_user: YWRtaW4=
    type: Opaque
  service:
    port: 80
    targetPort: 3000
    type: LoadBalancer
ingress:
  enabled: true
  prefix: /
  servicePort: 80
  virtual_host_fqdn: grafana.system.tanzu
```

Grafana-Konfiguration

Die Grafana-Konfiguration ist in `grafana-data-values.yaml` festgelegt. In der Tabelle sind die verfügbaren Parameter aufgeführt und beschrieben.

Tabelle 11-9. Grafana-Konfigurationsparameter

Parameter	Beschreibung	Typ	Standard
monitoring.namespace	Namespace, in dem Prometheus bereitgestellt wird	string	tanzu-system-monitoring
monitoring.create_namespace	Das Flag gibt an, ob der durch monitoring.namespace angegebene Namespace erstellt werden soll.	Boolean	false
monitoring.grafana.cluster_role.apiGroups	Für die Grafana-Clusterrolle definierte API-Gruppe	Liste	[""]
monitoring.grafana.cluster_role.resources	Für die Grafana-Clusterrolle definierte Ressourcen	Liste	["configmaps", "secrets"]
monitoring.grafana.cluster_role.verbs	Für die Clusterrolle definierte Zugriffsberechtigung	Liste	["get", "watch", "list"]
monitoring.grafana.config.grafana_ini	Details zur Grafana-Konfigurationsdatei	Konfigurationsdatei	grafana.ini In dieser Datei wird die `grafana_net`-URL für die Integration in Grafana verwendet, z. B. zum Importieren des Dashboards direkt von Grafana.com.
monitoring.grafana.config.datasource.type	Grafana-Datenquellentyp	string	Prometheus
monitoring.grafana.config.datasource.access	Zugriffsmodus: Proxy oder direkt (Server oder Browser auf der Benutzeroberfläche)	string	Proxy
monitoring.grafana.config.datasource.isDefault	Als Standard-Grafana-Datenquelle markieren	Boolean	true
monitoring.grafana.config.provider_yaml	Konfigurationsdatei zum Definieren des Grafana-Dashboardanbieters	yaml-Datei	provider.yaml
monitoring.grafana.service.type	Diensttyp, um Grafana verfügbar zu machen. Unterstützte Werte: ClusterIP, NodePort, LoadBalancer	string	vSphere: NodePort, aws/azure: LoadBalancer

Tabelle 11-9. Grafana-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
monitoring.grafana.pvc.storage_class	Definieren Sie den Zugriffsmodus für die Beanspruchung eines dauerhaften Volumes. Unterstützte Werte: ReadWriteOnce, ReadOnlyMany, ReadWriteMany	string	ReadWriteOnce
monitoring.grafana.cab.storage	Definition der Speichergröße für die Beanspruchung eines persistenten Volumes	string	2Gi
monitoring.grafana.deployment.replicas	Anzahl der Grafana-Replikate	integer	1
monitoring.grafana.image.repository	Speicherort des Repositorys mit dem Grafana-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg/grafana
monitoring.grafana.image.name	Name des Grafana-Images	string	Grafana
monitoring.grafana.image.tag	Grafana-Image-Tag. Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Version durchführen.	string	v7.3.5_vmware.1
monitoring.grafana.image.pullPolicy	Pull-Richtlinie für das Grafana-Image	string	IfNotPresent
monitoring.grafana.secret.type	Für das Grafana-Dashboard definierter geheimer Schlüsseltyp	string	Undurchsichtig
monitoring.grafana.secret.admin_user	Benutzername für den Zugriff auf das Grafana-Dashboard	string	YWRtaW4= Der Wert ist Base64-codiert; zum Entschlüsseln: `echo "xxxxxx" | base64 --decode`
monitoring.grafana.secret.admin_password	Kennwort für den Zugriff auf das Grafana-Dashboard	string	null
monitoring.grafana.secret.ldap_toml	Bei Verwendung von ldap auth, LDAP-Konfigurationsdateipfad	string	""

Tabelle 11-9. Grafana-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
monitoring.grafana_init_container.image.repository	Repository mit Grafana-Init-Container-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg/grafana
monitoring.grafana_init_container.image.name	Name des Grafana-Init-Container-Images	string	k8s-sidecar
monitoring.grafana_init_container.image.tag	Tag des Grafana-Init-Container-Images. Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Version durchführen.	string	0.1.99
monitoring.grafana_init_container.image.pullPolicy	Pull-Richtlinie für das Grafana-Init-Container-Image	string	IfNotPresent
monitoring.grafana_sc_dashboard.image.repository	Repository mit dem Grafana-Dashboard-Image. Als Standardwert wird die öffentliche VMware-Registrierung verwendet. Ändern Sie diesen Wert, wenn Sie ein privates Repository verwenden (z. B. Air-Gap-Umgebung).	string	projects.registry.vmware.com/tkg/grafana
monitoring.grafana_sc_dashboard.image.name	Name des Grafana-Dashboard-Images	string	k8s-sidecar
monitoring.grafana_sc_dashboard.image.tag	Tag des Grafana-Dashboard-Images. Dieser Wert muss möglicherweise aktualisiert werden, wenn Sie ein Upgrade der Version durchführen.	string	0.1.99
monitoring.grafana_sc_dashboard.image.pullPolicy	Pull-Richtlinie für das Grafana-Dashboard-Image	string	IfNotPresent
monitoring.grafana.ingress.enabled	Ingress für Grafana aktivieren/deaktivieren	Boolean	true
monitoring.grafana.ingress.virtual_host_fqdn	Hostname für den Zugriff auf Grafana	string	grafana.system.tanzu
monitoring.grafana.ingress.prefix	Pfadpräfix für Grafana	string	/

Tabelle 11-9. Grafana-Konfigurationsparameter (Fortsetzung)

Parameter	Beschreibung	Typ	Standard
monitoring.grafana.ingress.tlsCertificate.tls.crt	Optionales Zertifikat für Ingress, wenn Sie Ihr eigenes TLS-Zertifikat verwenden möchten. Standardmäßig wird ein selbstsigniertes Zertifikat generiert	string	Generiertes Zertifikat
monitoring.grafana.ingress.tlsCertificate.tls.key	Optionaler privater Zertifikatsschlüssel für Ingress, wenn Sie Ihr eigenes TLS-Zertifikat verwenden möchten.	string	Generierter Zertifikatsschlüssel

Referenz zum Harbor-Paket

Dieses Thema enthält Referenzinformationen für das Harbor Registry-Paket.

Informationen zu Harbor Registry

Harbor (https://goharbor.io/) ist ein Open Source-Containerregistrierungssystem, das ein Image-Repository, Image-Schwachstellenscans und Projektverwaltung bereitstellt.

Informationen zum Installieren des Harbor-Pakets in einem TKG-Cluster auf Supervisor finden Sie in den folgenden Themen:

- TKr für vSphere 8.x: Installieren der Harbor-Registrierung
- TKr für vSphere 7.x: #unique_178

Harbor-Komponenten

Das Harbor-Paket installiert die beiden in der Tabelle aufgeführten Container im Cluster. Das Paket ruft die Container aus der öffentlichen Registrierung ab, die im Paket-Repository angegeben ist.

Container	Ressourcentyp	Replikate	Beschreibung
`harbor-core`	Bereitstellung	1	Verwaltungs- und Konfigurationsserver für Envoy
`harbor-database`	Pod	1	Postgres-Datenbank
`harbor-jobservice`	Bereitstellung	1	Harbor-Auftragsdienst
`harbor-notary-server`	Bereitstellung	1	Harbor-Notariatsdienst
`harbor-notary-signer`	Bereitstellung	1	Harbor-Notariat
`harbor-portal`	Bereitstellung	1	Harbor-Webschnittstelle
`harbor-redis`	Pod	1	Harbor-Redis-Instanz

Container	Ressourcentyp	Replikate	Beschreibung
`harbor-registry`	Bereitstellung	2	Harbor-Containerregistrierungsinstanz
`harbor-trivy`	Pod	1	Schwachstellen-Scanner für Harbor-Image

Harbor-Datenwerte

Unten finden Sie ein Beispiel für `harbor-data-values` für die Installation von Harbor.

Datenwert	Beschreibung
`hostname: myharbordomain.com`	Der FQDN für den Zugriff auf die Harbor-Verwaltungsbenutzeroberfläche und den Registrierungsdienst.
`harborAdminPassword: change-it`	Das anfängliche Kennwort für das Harbor-Administratorkonto. Dies wird nur während der Installation angewendet. Sie können es nach der Installation über die Harbor-Benutzeroberfläche oder -API aktualisieren.
`secretKey: 0123456789ABCDEF`	Der für die Verschlüsselung verwendete geheime Schlüssel. Muss eine Folge von 16 Zeichen sein.
`database.password: change-it`	Das anfängliche Kennwort der Postgres-Datenbank.
`core.secret: change-it`	Der geheime Schlüssel wird verwendet, wenn der Kernserver mit anderen Komponenten kommuniziert.
`xsrfKey: 0123456789ABCDEF0123456789ABCDEF`	Der XSRF-Schlüssel. Muss eine Folge von 32 Zeichen sein.
`jobservice.secret: change-it`	Der geheime Schlüssel wird verwendet, wenn der Auftragsdienst mit anderen Komponenten kommuniziert.
`registry.secret: change-it`	Der geheime Schlüssel wird verwendet, um den Upload-Status vom Client- und Registrierungsspeicher-Backend zu sichern.
`persistence.persistentVolumeClaim.registry.storageClass: mystorageclass`	Geben Sie die vSphere Speicherrichtlinie an, die für die Bereitstellung des Volumes verwendet wird.
`persistence.persistentVolumeClaim.jobservice.storageClass: mystorageclass`	Geben Sie die vSphere Speicherrichtlinie an, die für die Bereitstellung des Volumes verwendet wird.
`persistence.persistentVolumeClaim.database.storageClass: mystorageclass`	Geben Sie die vSphere Speicherrichtlinie an, die für die Bereitstellung des Volumes verwendet wird.
`persistence.persistentVolumeClaim.redis.storageClass: mystorageclass`	Geben Sie die vSphere Speicherrichtlinie an, die für die Bereitstellung des Volumes verwendet wird.
`persistence.persistentVolumeClaim.trivy.storageClass: mystorageclass`	Geben Sie die vSphere Speicherrichtlinie an, die für die Bereitstellung des Volumes verwendet wird.

Harbor-Konfiguration

Die Harbor-Konfiguration wird in der `harbor-data-values.yaml`-Datei festgelegt. In der Tabelle werden die mindestens erforderlichen Felder für die Bereitstellung aufgelistet und beschrieben.

Eigenschaft	Wert	Beschreibung
Hostname	FQDN	Der FQDN, den Sie für den Zugriff auf die Benutzeroberfläche von Harbor und für die Referenzierung der Registrierung in Clientanwendungen festgelegt haben. Die Domäne sollte in einem externen DNS-Server so konfiguriert werden, dass sie in die von Contour erstellte Envoy-Dienst-IP aufgelöst wird.
tlsCertificate.tlsSecretLabels	{"managed-by": "vmware-vRegistry"}	Das Zertifikat, das Tanzu Kubernetes Grid verwendet, um das Harbor-CA-Zertifikat als vertrauenswürdigen Root auf Tanzu Kubernetes Grid-Clustern zu installieren.
persistence.persistentVolumeClaim.registry.storageClass	Der Name einer Speicherrichtlinie.	Eine Speicherklasse, die für die Harbor-PVCs für die Registrierung verwendet wird.
persistence.persistentVolumeClaim.jobservice.storageClass	Der Name einer Speicherrichtlinie.	Eine Speicherklasse, die für die Harbor-PVCs für Jobservices verwendet wird.
persistence.persistentVolumeClaim.database.storageClass	Der Name einer Speicherrichtlinie.	Eine Speicherklasse, die für die Harbor-Datenbanken-PVCs verwendet wird.
persistence.persistentVolumeClaim.redis.storageClass	Der Name einer Speicherrichtlinie.	Eine Speicherklasse, die für die Harbor-PVCs für Redis verwendet wird.
persistence.persistentVolumeClaim.trivy.storageClass	Der Name einer Speicherrichtlinie.	Eine Speicherklasse, die für Harbor-PVCs für Trivy verwendet wird.

Installieren von Standardpaketen in einem TKG-Cluster mithilfe von TKr für vSphere 7.x

In diesem Abschnitt finden Sie Informationen zum Installieren unterstützter Standardpakete in TKG-Dienst-Clustern, die mit unterstützten TKrs für vSphere 7.x bereitgestellt werden.

Workflow zum Installieren von Standardpaketen auf TKr für vSphere 7.x

Dieser Abschnitt enthält Anweisungen zum Installieren von Standardpaketen auf TKG-Clustern, die mit TKr für vSphere 7.x bereitgestellt sind.

Anforderungen

Diese Anweisungen werden mit TKr v1.27.10 für vSphere 7.0.3.6 und TKr v1.27.10 für vSphere 8.0.1.1 validiert. Zum Zeitpunkt der Veröffentlichung war dies die zuletzt verfügbare TKr für vSphere 7.x. TKrs für vSphere 7.x können auf vSphere 8.x ausgeführt werden, um vSphere 7.x auf vSphere 8.x zu aktualisieren.

Beachten Sie die folgenden Voraussetzungen:

- Arbeitslastverwaltung aktiviert
- Supervisor-bereitgestellt
- vSphere-Namespace-erstellt

  Weitere Informationen hierzu finden Sie unter Erstellen eines vSphere-Namespace für das Hosting von TKG-Dienst-Clustern.
- Linux-Client mit installierten Kubernetes-CLI-Tools für vSphere

  Weitere Informationen hierzu finden Sie unter Installieren des Kubernetes-CLI-Tools für vSphere.

Hinweis Wenn Sie einen TKG-Cluster verwenden, der mit einer TKr für vSphere 8.x bereitgestellt wurde, finden Sie in der folgenden Dokumentation Anweisungen zur Installation von Standardpaketen: Installieren von Standardpaketen auf einem TKG-Cluster mithilfe der TKr für vSphere 8.x. Weitere Informationen zu den TKr-Versionen finden Sie in den Versionshinweisen.

Erstellen eines TKG-Clusters

Erstellen Sie einen TKG-Cluster zum Hosten von Standardpaketen.

1 Erstellen Sie einen TKG-Cluster.

  Weitere Informationen hierzu finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl.

  Beispiel einer Clusterspezifikation für die Photon-Edition von TKr v1.27.10.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkgs-cluster-photon
  namespace: tkgs-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: vsan-esa-default-policy-raid5
      tkr:
        reference:
          name: v1.27.10---vmware.1-fips.1-tkg.1 #TKR for v7
    nodePools:
    - name: worker
      replicas: 3
      vmClass: guaranteed-medium
```

```
      storageClass: vsan-esa-default-policy-raid5
  settings:
    storage:
      defaultClass: vsan-esa-default-policy-raid5
```

Beispiel einer Clusterspezifikation für die Ubuntu-Edition von TKr v1.27.10.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkgs-cluster-ubuntu
  namespace: tkgs-ns
  annotations:
    run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: vsan-esa-default-policy-raid5
      tkr:
        reference:
          name: v1.27.10---vmware.1-fips.1-tkg.1.ubuntu #TKR for v7
    nodePools:
    - name: worker
      replicas: 3
      vmClass: guaranteed-medium
      storageClass: vsan-esa-default-policy-raid5
  settings:
    storage:
      defaultClass: vsan-esa-default-policy-raid5
```

Installieren des Carvel-Paketmanagers

Installieren Sie den Carvel-Paketmanager.

1 Melden Sie sich beim TKG-Cluster an.

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USER@vsphere.local --tanzu-
kubernetes-cluster-name tkgs-cluster-photon --tanzu-kubernetes-cluster-namespace tkgs-ns
```

2 Installieren Sie den Carvel-Paketmanager.

```
wget -O- https://carvel.dev/install.sh > install.sh
```

```
sed -i 's/wget -nv -O-/wget --no-check-certificate -nv -O-/' install.sh
```

```
sudo bash install.sh
```

3 Überprüfen Sie die Installation.

```
imgpkg version
```

Installieren des Kapp-Controllers

Weitere Informationen hierzu finden Sie unter Installieren des Kapp-Controllers auf TKr für vSphere 7.x.

Hinzufügen eines Paketrepositorys

Fügen Sie die gewünschte Version für das Paketrepository hinzu.

1 Listen Sie das aktuelle Repository-Tag auf.

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

2 Erstellen Sie `packagerepo.yaml`.

Aktualisieren Sie die Repository-Version so, dass sie mit der Zielversion übereinstimmt.

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageRepository
metadata:
  name: tanzu-standard
  namespace: tkg-system
spec:
  fetch:
    imgpkgBundle:
      image: projects.registry.vmware.com/tkg/packages/standard/repo:v2024.2.1
```

3 Installieren Sie das Paketrepository.

```
kubectl apply -f packagerepo.yaml
```

Erwartetes Ergebnis:

```
packagerepository.packaging.carvel.dev/tanzu-standard created
```

4 Überprüfen Sie das Paketrepository.

```
kubectl get packagerepositories -A
```

Erwartetes Ergebnis:

```
NAMESPACE    NAME             AGE    DESCRIPTION
tkg-system   tanzu-standard   3m9s   Reconcile succeeded
```

Installieren des Zertifikatmanagers

Weitere Informationen hierzu finden Sie unter Installieren von Cert Manager auf TKr für vSphere 7.x.

Installieren von Contour mit Envoy

Weitere Informationen hierzu finden Sie unter Installieren von Contour auf TKr für vSphere 7.x.

Installieren von ExternalDNS

Weitere Informationen hierzu finden Sie unter Installieren von ExternalDNS auf TKr für vSphere 7.x.

Installieren von Fluent Bit für die Protokollweiterleitung

Weitere Informationen hierzu finden Sie unter Installieren von Fluent Bit auf TKr für vSphere 7.x.

Installieren von Prometheus

Weitere Informationen hierzu finden Sie unter Installieren von Prometheus auf TKr für vSphere 7.x.

Installieren von Grafana

Weitere Informationen hierzu finden Sie unter Installieren von Grafana auf TKr für vSphere 7.x.

Installieren von Harbor

Weitere Informationen hierzu finden Sie unter Installieren von Harbor auf TKr für vSphere 7.x.

Installieren des Kapp-Controllers auf TKr für vSphere 7.x

Lesen Sie diese Anweisungen zum Installieren des Kapp-Controllers auf einem TKG-Cluster, der mit TKr für vSphere 7.x bereitgestellt wurde.

Voraussetzungen

Weitere Informationen hierzu finden Sie unter Workflow zum Installieren von Standardpaketen auf TKr für vSphere 7.x.

Installieren des Kapp-Controllers

Wichtig Diese Anweisungen gelten speziell für TKrs für vSphere 7.x. TKrs für vSphere 8.x enthalten bereits das Kapp-Controller-Paket. Installieren Sie den Kapp-Controller nicht manuell auf einer TKr für vSphere 8.x.

Installieren Sie den Kapp-Controller.

1 Erstellen Sie eine Bindung zum Ausführen des Kapp-Controller-Pods.

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding --
clusterrole=cluster-admin --group=system:authenticated
```

Erwartetes Ergebnis:

```
clusterrolebinding.rbac.authorization.k8s.io/default-tkg-admin-privileged-binding created
```

2 Bereiten Sie die Datei `kapp-controller.yaml` vor.

Informationen hierzu finden Sie unter

3 Installieren Sie den Kapp-Controller.

```
kubectl apply -f kapp-controller.yaml
```

4 Überprüfen Sie die Installation des Kapp-Controllers.

```
kubectl get all -n tkg-system
```

Beispielergebnis:

```
NAME                                 READY   STATUS    RESTARTS   AGE
pod/kapp-controller-b7576ddd-p8s87   2/2     Running   0          5m33s
NAME                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/packaging-api    ClusterIP   198.201.96.77   <none>        443/TCP   5m34s
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/kapp-controller     1/1     1            1           5m33s
```

5 Überprüfen Sie die benutzerdefinierte Carvel-Ressource.

```
kubectl get crd | grep carvel
```

Beispielergebnis:

```
internalpackagemetadatas.internal.packaging.carvel.dev   2024-03-12T08:27:21Z
internalpackages.internal.packaging.carvel.dev           2024-03-12T08:27:21Z
packageinstalls.packaging.carvel.dev                     2024-03-12T08:27:21Z
packagerepositories.packaging.carvel.dev                 2024-03-12T08:27:22Z
```

kapp-controller.yaml

Die folgende `kapp-controller.yaml` enthält die erforderlichen `securityContext`-Einstellungen.

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: tkg-system
---
apiVersion: v1
kind: Namespace
metadata:
  name: kapp-controller-packaging-global
---
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  name: v1alpha1.data.packaging.carvel.dev
spec:
  group: data.packaging.carvel.dev
  groupPriorityMinimum: 100
  service:
    name: packaging-api
    namespace: tkg-system
```

```
  version: v1alpha1
  versionPriority: 100
---
apiVersion: v1
kind: Service
metadata:
  name: packaging-api
  namespace: tkg-system
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: api
  selector:
    app: kapp-controller
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: internalpackagemetadatas.internal.packaging.carvel.dev
spec:
  group: internal.packaging.carvel.dev
  names:
    kind: InternalPackageMetadata
    listKind: InternalPackageMetadataList
    plural: internalpackagemetadatas
    singular: internalpackagemetadata
  scope: Namespaced
  versions:
  - name: v1alpha1
    schema:
      openAPIV3Schema:
        properties:
          apiVersion:
            description: 'APIVersion defines the versioned schema of this representation
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources'
            type: string
          kind:
            description: 'Kind is a string value representing the REST resource this
              object represents. Servers may infer this from the endpoint the client
              submits requests to. Cannot be updated. In CamelCase. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds'
            type: string
          metadata:
            type: object
          spec:
            properties:
              categories:
                description: Classifiers of the package (optional; Array of strings)
                items:
                  type: string
                type: array
              displayName:
```

```
                description: Human friendly name of the package (optional; string)
                type: string
              iconSVGBase64:
                description: Base64 encoded icon (optional; string)
                type: string
              longDescription:
                description: Long description of the package (optional; string)
                type: string
              maintainers:
                description: List of maintainer info for the package. Currently only
                  supports the name key. (optional; array of maintner info)
                items:
                  properties:
                    name:
                      type: string
                  type: object
                type: array
              providerName:
                description: Name of the entity distributing the package (optional;
                  string)
                type: string
              shortDescription:
                description: Short desription of the package (optional; string)
                type: string
              supportDescription:
                description: Description of the support available for the package
                  (optional; string)
                type: string
            type: object
        required:
        - spec
        type: object
    served: true
    storage: true
    subresources:
      status: {}
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: internalpackages.internal.packaging.carvel.dev
spec:
  group: internal.packaging.carvel.dev
  names:
    kind: InternalPackage
    listKind: InternalPackageList
    plural: internalpackages
    singular: internalpackage
  scope: Namespaced
  versions:
  - name: v1alpha1
    schema:
      openAPIV3Schema:
        properties:
          apiVersion:
```

```
            description: 'APIVersion defines the versioned schema of this representation
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources'
            type: string
          kind:
            description: 'Kind is a string value representing the REST resource this
              object represents. Servers may infer this from the endpoint the client
              submits requests to. Cannot be updated. In CamelCase. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds'
            type: string
          metadata:
            type: object
          spec:
            properties:
              capacityRequirementsDescription:
                description: 'System requirements needed to install the package. Note:
                  these requirements will not be verified by kapp-controller on installation.
                  (optional; string)'
                type: string
              includedSoftware:
                description: IncludedSoftware can be used to show the software contents
                  of a Package. This is especially useful if the underlying versions
                  do not match the Package version
                items:
                  description: IncludedSoftware contains the underlying Software Contents
                    of a Package
                  properties:
                    description:
                      type: string
                    displayName:
                      type: string
                    version:
                      type: string
                  type: object
                type: array
              kappControllerVersionSelection:
                description: KappControllerVersionSelection specifies the versions
                  of kapp-controller which can install this package
                properties:
                  constraints:
                    type: string
                type: object
              kubernetesVersionSelection:
                description: KubernetesVersionSelection specifies the versions of
                  k8s which this package can be installed on
                properties:
                  constraints:
                    type: string
                type: object
              licenses:
                description: Description of the licenses that apply to the package
                  software (optional; Array of strings)
                items:
                  type: string
```

```
      type: array
    refName:
      description: The name of the PackageMetadata associated with this
        version Must be a valid PackageMetadata name (see PackageMetadata
        CR for details) Cannot be empty
      type: string
    releaseNotes:
      description: Version release notes (optional; string)
      type: string
    releasedAt:
      description: Timestamp of release (iso8601 formatted string; optional)
      format: date-time
      nullable: true
      type: string
    template:
      properties:
        spec:
          properties:
            canceled:
              description: Cancels current and future reconciliations (optional;
                default=false)
              type: boolean
            cluster:
              description: Specifies that app should be deployed to destination
                cluster; by default, cluster is same as where this resource
                resides (optional; v0.5.0+)
              properties:
                kubeconfigSecretRef:
                  description: Specifies secret containing kubeconfig (required)
                  properties:
                    key:
                      description: Specifies key that contains kubeconfig
                        (optional)
                      type: string
                    name:
                      description: Specifies secret name within app's namespace
                        (required)
                      type: string
                  type: object
                namespace:
                  description: Specifies namespace in destination cluster
                    (optional)
                  type: string
              type: object
            deploy:
              items:
                properties:
                  kapp:
                    description: Use kapp to deploy resources
                    properties:
                      delete:
                        description: Configuration for delete command (optional)
                        properties:
                          rawOptions:
                            description: Pass through options to kapp delete
```

```
                                (optional)
                              items:
                                type: string
                              type: array
                          type: object
                        inspect:
                          description: 'Configuration for inspect command
                            (optional) as of kapp-controller v0.31.0, inspect
                            is disabled by default add rawOptions or use an
                            empty inspect config like `inspect: {}` to enable'
                          properties:
                            rawOptions:
                              description: Pass through options to kapp inspect
                                (optional)
                              items:
                                type: string
                              type: array
                          type: object
                        intoNs:
                          description: Override namespace for all resources
                            (optional)
                          type: string
                        mapNs:
                          description: Provide custom namespace override mapping
                            (optional)
                          items:
                            type: string
                          type: array
                        rawOptions:
                          description: Pass through options to kapp deploy
                            (optional)
                          items:
                            type: string
                          type: array
                      type: object
                  type: object
                type: array
              fetch:
                items:
                  properties:
                    git:
                      description: Uses git to clone repository
                      properties:
                        lfsSkipSmudge:
                          description: Skip lfs download (optional)
                          type: boolean
                        ref:
                          description: Branch, tag, commit; origin is the
                            name of the remote (optional)
                          type: string
                        refSelection:
                          description: Specifies a strategy to resolve to
                            an explicit ref (optional; v0.24.0+)
                          properties:
                            semver:
```

```
            properties:
              constraints:
                type: string
              prereleases:
                properties:
                  identifiers:
                    items:
                      type: string
                    type: array
                type: object
            type: object
        type: object
      secretRef:
        description: 'Secret with auth details. allowed
          keys: ssh-privatekey, ssh-knownhosts, username,
          password (optional) (if ssh-knownhosts is not
          specified, git will not perform strict host checking)'
        properties:
          name:
            description: Object is expected to be within
              same namespace
            type: string
        type: object
      subPath:
        description: Grab only portion of repository (optional)
        type: string
      url:
        description: http or ssh urls are supported (required)
        type: string
    type: object
  helmChart:
    description: Uses helm fetch to fetch specified chart
    properties:
      name:
        description: 'Example: stable/redis'
        type: string
      repository:
        properties:
          secretRef:
            properties:
              name:
                description: Object is expected to be within
                  same namespace
                type: string
            type: object
          url:
            description: Repository url; scheme of oci://
              will fetch experimental helm oci chart (v0.19.0+)
              (required)
            type: string
        type: object
      version:
        type: string
    type: object
  http:
```

```
      description: Uses http library to fetch file
      properties:
        secretRef:
          description: 'Secret to provide auth details (optional)
            Secret may include one or more keys: username,
            password'
          properties:
            name:
              description: Object is expected to be within
                same namespace
              type: string
          type: object
        sha256:
          description: Checksum to verify after download (optional)
          type: string
        subPath:
          description: Grab only portion of download (optional)
          type: string
        url:
          description: 'URL can point to one of following
            formats: text, tgz, zip http and https url are
            supported; plain file, tgz and tar types are supported
            (required)'
          type: string
      type: object
    image:
      description: Pulls content from Docker/OCI registry
      properties:
        secretRef:
          description: 'Secret may include one or more keys:
            username, password, token. By default anonymous
            access is used for authentication.'
          properties:
            name:
              description: Object is expected to be within
                same namespace
              type: string
          type: object
        subPath:
          description: Grab only portion of image (optional)
          type: string
        tagSelection:
          description: Specifies a strategy to choose a tag
            (optional; v0.24.0+) if specified, do not include
            a tag in url key
          properties:
            semver:
              properties:
                constraints:
                  type: string
                prereleases:
                  properties:
                    identifiers:
                      items:
                        type: string
```

```
                                type: array
                            type: object
                        type: object
                    type: object
                  url:
                    description: 'Docker image url; unqualified, tagged,
                      or digest references supported (required) Example:
                      username/app1-config:v0.1.0'
                    type: string
                type: object
              imgpkgBundle:
                description: Pulls imgpkg bundle from Docker/OCI registry
                  (v0.17.0+)
                properties:
                  image:
                    description: Docker image url; unqualified, tagged,
                      or digest references supported (required)
                    type: string
                  secretRef:
                    description: 'Secret may include one or more keys:
                      username, password, token. By default anonymous
                      access is used for authentication.'
                    properties:
                      name:
                        description: Object is expected to be within
                          same namespace
                        type: string
                    type: object
                  tagSelection:
                    description: Specifies a strategy to choose a tag
                      (optional; v0.24.0+) if specified, do not include
                      a tag in url key
                    properties:
                      semver:
                        properties:
                          constraints:
                            type: string
                          prereleases:
                            properties:
                              identifiers:
                                items:
                                  type: string
                                type: array
                            type: object
                        type: object
                    type: object
                type: object
              inline:
                description: Pulls content from within this resource;
                  or other resources in the cluster
                properties:
                  paths:
                    additionalProperties:
                      type: string
                    description: Specifies mapping of paths to their
```

```
                    content; not recommended for sensitive values
                    as CR is not encrypted (optional)
                  type: object
                pathsFrom:
                  description: Specifies content via secrets and config
                    maps; data values are recommended to be placed
                    in secrets (optional)
                  items:
                    properties:
                      configMapRef:
                        properties:
                          directoryPath:
                            description: Specifies where to place
                              files found in secret (optional)
                            type: string
                          name:
                            type: string
                        type: object
                      secretRef:
                        properties:
                          directoryPath:
                            description: Specifies where to place
                              files found in secret (optional)
                            type: string
                          name:
                            type: string
                        type: object
                    type: object
                  type: array
              type: object
            path:
              description: Relative path to place the fetched artifacts
              type: string
          type: object
        type: array
      noopDelete:
        description: Deletion requests for the App will result in
          the App CR being deleted, but its associated resources will
          not be deleted (optional; default=false; v0.18.0+)
        type: boolean
      paused:
        description: Pauses _future_ reconciliation; does _not_ affect
          currently running reconciliation (optional; default=false)
        type: boolean
      serviceAccountName:
        description: Specifies that app should be deployed authenticated
          via given service account, found in this namespace (optional;
          v0.6.0+)
        type: string
      syncPeriod:
        description: Specifies the length of time to wait, in time
          + unit format, before reconciling. Always >= 30s. If value
          below 30s is specified, 30s will be used. (optional; v0.9.0+;
          default=30s)
        type: string
```

```
                    template:
                      items:
                        properties:
                          cue:
                            properties:
                              inputExpression:
                                description: Cue expression for single path component,
                                  can be used to unify ValuesFrom into a given field
                                  (optional)
                                type: string
                              outputExpression:
                                description: Cue expression to output, default will
                                  export all visible fields (optional)
                                type: string
                              paths:
                                description: Explicit list of files/directories
                                  (optional)
                                items:
                                  type: string
                                type: array
                              valuesFrom:
                                description: Provide values (optional)
                                items:
                                  properties:
                                    configMapRef:
                                      properties:
                                        name:
                                          type: string
                                      type: object
                                    downwardAPI:
                                      properties:
                                        items:
                                          items:
                                            properties:
                                              fieldPath:
                                                description: 'Required: Selects
                                                  a field of the app: only annotations,
                                                  labels, uid, name and namespace
                                                  are supported.'
                                                type: string
                                              kappControllerVersion:
                                                description: 'Optional: Get running
                                                  KappController version, defaults
                                                  (empty) to retrieving the current
                                                  running version.. Can be manually
                                                  supplied instead.'
                                                properties:
                                                  version:
                                                    type: string
                                                type: object
                                              kubernetesAPIs:
                                                description: 'Optional: Get running
                                                  KubernetesAPIs from cluster, defaults
                                                  (empty) to retrieving the APIs
                                                  from the cluster. Can be manually
```

```
                                        supplied instead, e.g ["group/version",
                                        "group2/version2"]'
                                      properties:
                                        groupVersions:
                                          items:
                                            type: string
                                          type: array
                                      type: object
                                    kubernetesVersion:
                                      description: 'Optional: Get running
                                        Kubernetes version from cluster,
                                        defaults (empty) to retrieving
                                        the version from the cluster.
                                        Can be manually supplied instead.'
                                      properties:
                                        version:
                                          type: string
                                      type: object
                                    name:
                                      type: string
                                  type: object
                                type: array
                              type: object
                            path:
                              type: string
                            secretRef:
                              properties:
                                name:
                                  type: string
                              type: object
                          type: object
                        type: array
                      type: object
                    helmTemplate:
                      description: Use helm template command to render helm
                        chart
                      properties:
                        kubernetesAPIs:
                          description: 'Optional: Use kubernetes group/versions
                            resources available in the live cluster'
                          properties:
                            groupVersions:
                              items:
                                type: string
                              type: array
                          type: object
                        kubernetesVersion:
                          description: 'Optional: Get Kubernetes version,
                            defaults (empty) to retrieving the version from
                            the cluster. Can be manually overridden to a value
                            instead.'
                          properties:
                            version:
                              type: string
                          type: object
```

```
                                name:
                                  description: Set name explicitly, default is App
                                    CR's name (optional; v0.13.0+)
                                  type: string
                                namespace:
                                  description: Set namespace explicitly, default is
                                    App CR's namespace (optional; v0.13.0+)
                                  type: string
                                path:
                                  description: Path to chart (optional; v0.13.0+)
                                  type: string
                                valuesFrom:
                                  description: One or more secrets, config maps, paths
                                    that provide values (optional)
                                  items:
                                    properties:
                                      configMapRef:
                                        properties:
                                          name:
                                            type: string
                                        type: object
                                      downwardAPI:
                                        properties:
                                          items:
                                            items:
                                              properties:
                                                fieldPath:
                                                  description: 'Required: Selects
                                                    a field of the app: only annotations,
                                                    labels, uid, name and namespace
                                                    are supported.'
                                                  type: string
                                                kappControllerVersion:
                                                  description: 'Optional: Get running
                                                    KappController version, defaults
                                                    (empty) to retrieving the current
                                                    running version.. Can be manually
                                                    supplied instead.'
                                                  properties:
                                                    version:
                                                      type: string
                                                  type: object
                                                kubernetesAPIs:
                                                  description: 'Optional: Get running
                                                    KubernetesAPIs from cluster, defaults
                                                    (empty) to retrieving the APIs
                                                    from the cluster. Can be manually
                                                    supplied instead, e.g ["group/version",
                                                    "group2/version2"]'
                                                  properties:
                                                    groupVersions:
                                                      items:
                                                        type: string
                                                      type: array
                                                  type: object
```

```
                                    kubernetesVersion:
                                      description: 'Optional: Get running
                                        Kubernetes version from cluster,
                                        defaults (empty) to retrieving
                                        the version from the cluster.
                                        Can be manually supplied instead.'
                                      properties:
                                        version:
                                          type: string
                                      type: object
                                    name:
                                      type: string
                                  type: object
                              type: array
                        type: object
                    path:
                      type: string
                    secretRef:
                      properties:
                        name:
                          type: string
                      type: object
                type: object
            type: array
      type: object
    jsonnet:
      description: TODO implement jsonnet
      type: object
    kbld:
      description: Use kbld to resolve image references to
        use digests
      properties:
        paths:
          items:
            type: string
          type: array
      type: object
    kustomize:
      description: TODO implement kustomize
      type: object
    sops:
      description: Use sops to decrypt *.sops.yml files (optional;
        v0.11.0+)
      properties:
        age:
          properties:
            privateKeysSecretRef:
              description: Secret with private armored PGP
                private keys (required)
              properties:
                name:
                  type: string
              type: object
          type: object
        paths:
```

```
                                  description: Lists paths to decrypt explicitly (optional;
                                    v0.13.0+)
                                  items:
                                    type: string
                                  type: array
                                pgp:
                                  description: Use PGP to decrypt files (required)
                                  properties:
                                    privateKeysSecretRef:
                                      description: Secret with private armored PGP
                                        private keys (required)
                                      properties:
                                        name:
                                          type: string
                                      type: object
                                  type: object
                              type: object
                            ytt:
                              description: Use ytt to template configuration
                              properties:
                                fileMarks:
                                  description: Control metadata about input files
                                    passed to ytt (optional; v0.18.0+) see https://
carvel.dev/ytt/docs/latest/file-marks/
                                      for more details
                                  items:
                                    type: string
                                  type: array
                                ignoreUnknownComments:
                                  description: Ignores comments that ytt doesn't recognize
                                    (optional; default=false)
                                  type: boolean
                                inline:
                                  description: Specify additional files, including
                                    data values (optional)
                                  properties:
                                    paths:
                                      additionalProperties:
                                        type: string
                                      description: Specifies mapping of paths to their
                                        content; not recommended for sensitive values
                                        as CR is not encrypted (optional)
                                      type: object
                                    pathsFrom:
                                      description: Specifies content via secrets and
                                        config maps; data values are recommended to
                                        be placed in secrets (optional)
                                      items:
                                        properties:
                                          configMapRef:
                                            properties:
                                              directoryPath:
                                                description: Specifies where to place
                                                  files found in secret (optional)
                                                type: string
```

```
                                        name:
                                          type: string
                                      type: object
                                    secretRef:
                                      properties:
                                        directoryPath:
                                          description: Specifies where to place
                                            files found in secret (optional)
                                          type: string
                                        name:
                                          type: string
                                      type: object
                                  type: object
                              type: array
                          type: object
                        paths:
                          description: Lists paths to provide to ytt explicitly
                            (optional)
                          items:
                            type: string
                          type: array
                        strict:
                          description: Forces strict mode https://github.com/k14s/ytt/
blob/develop/docs/strict.md
                            (optional; default=false)
                          type: boolean
                        valuesFrom:
                          description: Provide values via ytt's --data-values-file
                            (optional; v0.19.0-alpha.9)
                          items:
                            properties:
                              configMapRef:
                                properties:
                                  name:
                                    type: string
                                type: object
                              downwardAPI:
                                properties:
                                  items:
                                    items:
                                      properties:
                                        fieldPath:
                                          description: 'Required: Selects
                                            a field of the app: only annotations,
                                            labels, uid, name and namespace
                                            are supported.'
                                          type: string
                                        kappControllerVersion:
                                          description: 'Optional: Get running
                                            KappController version, defaults
                                            (empty) to retrieving the current
                                            running version.. Can be manually
                                            supplied instead.'
                                          properties:
                                            version:
```

```
                                        type: string
                                    type: object
                                  kubernetesAPIs:
                                    description: 'Optional: Get running
                                      KubernetesAPIs from cluster, defaults
                                      (empty) to retrieving the APIs
                                      from the cluster. Can be manually
                                      supplied instead, e.g ["group/version",
                                      "group2/version2"]'
                                    properties:
                                      groupVersions:
                                        items:
                                          type: string
                                        type: array
                                    type: object
                                  kubernetesVersion:
                                    description: 'Optional: Get running
                                      Kubernetes version from cluster,
                                      defaults (empty) to retrieving
                                      the version from the cluster.
                                      Can be manually supplied instead.'
                                    properties:
                                      version:
                                        type: string
                                    type: object
                                  name:
                                    type: string
                                type: object
                              type: array
                            type: object
                          path:
                            type: string
                          secretRef:
                            properties:
                              name:
                                type: string
                            type: object
                        type: object
                      type: array
                    type: object
                type: object
              type: array
          type: object
        required:
        - spec
        type: object
      valuesSchema:
        description: valuesSchema can be used to show template values that
          can be configured by users when a Package is installed in an OpenAPI
          schema format.
        properties:
          openAPIv3:
            nullable: true
            type: object
            x-kubernetes-preserve-unknown-fields: true
```

```
                type: object
              version:
                description: Package version; Referenced by PackageInstall; Must be
                  valid semver (required) Cannot be empty
                type: string
            type: object
        required:
        - spec
        type: object
    served: true
    storage: true
    subresources:
      status: {}
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: apps.kappctrl.k14s.io
spec:
  group: kappctrl.k14s.io
  names:
    categories:
    - carvel
    kind: App
    listKind: AppList
    plural: apps
    singular: app
  scope: Namespaced
  versions:
  - additionalPrinterColumns:
    - description: Friendly description
      jsonPath: .status.friendlyDescription
      name: Description
      type: string
    - description: Last time app started being deployed. Does not mean anything was
        changed.
      jsonPath: .status.deploy.startedAt
      name: Since-Deploy
      type: date
    - description: Time since creation
      jsonPath: .metadata.creationTimestamp
      name: Age
      type: date
    name: v1alpha1
    schema:
      openAPIV3Schema:
        description: 'An App is a set of Kubernetes resources. These resources could
          span any number of namespaces or could be cluster-wide (e.g. CRDs). An App
          is represented in kapp-controller using a App CR. The App CR comprises of
          three main sections: spec.fetch â€" declare source for fetching configuration
          and OCI images spec.template â€" declare templating tool and values spec.deploy
          â€" declare deployment tool and any deploy specific configuration'
        properties:
          apiVersion:
            description: 'APIVersion defines the versioned schema of this representation
```

```
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources'
            type: string
          kind:
            description: 'Kind is a string value representing the REST resource this
              object represents. Servers may infer this from the endpoint the client
              submits requests to. Cannot be updated. In CamelCase. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds'
            type: string
          metadata:
            type: object
          spec:
            properties:
              canceled:
                description: Cancels current and future reconciliations (optional;
                  default=false)
                type: boolean
              cluster:
                description: Specifies that app should be deployed to destination
                  cluster; by default, cluster is same as where this resource resides
                  (optional; v0.5.0+)
                properties:
                  kubeconfigSecretRef:
                    description: Specifies secret containing kubeconfig (required)
                    properties:
                      key:
                        description: Specifies key that contains kubeconfig (optional)
                        type: string
                      name:
                        description: Specifies secret name within app's namespace
                          (required)
                        type: string
                    type: object
                  namespace:
                    description: Specifies namespace in destination cluster (optional)
                    type: string
                type: object
              deploy:
                items:
                  properties:
                    kapp:
                      description: Use kapp to deploy resources
                      properties:
                        delete:
                          description: Configuration for delete command (optional)
                          properties:
                            rawOptions:
                              description: Pass through options to kapp delete (optional)
                              items:
                                type: string
                              type: array
                          type: object
                        inspect:
                          description: 'Configuration for inspect command (optional)
```

```
                    as of kapp-controller v0.31.0, inspect is disabled by
                    default add rawOptions or use an empty inspect config
                    like `inspect: {}` to enable'
                  properties:
                    rawOptions:
                      description: Pass through options to kapp inspect (optional)
                      items:
                        type: string
                      type: array
                  type: object
                intoNs:
                  description: Override namespace for all resources (optional)
                  type: string
                mapNs:
                  description: Provide custom namespace override mapping (optional)
                  items:
                    type: string
                  type: array
                rawOptions:
                  description: Pass through options to kapp deploy (optional)
                  items:
                    type: string
                  type: array
              type: object
          type: object
        type: array
      fetch:
        items:
          properties:
            git:
              description: Uses git to clone repository
              properties:
                lfsSkipSmudge:
                  description: Skip lfs download (optional)
                  type: boolean
                ref:
                  description: Branch, tag, commit; origin is the name of
                    the remote (optional)
                  type: string
                refSelection:
                  description: Specifies a strategy to resolve to an explicit
                    ref (optional; v0.24.0+)
                  properties:
                    semver:
                      properties:
                        constraints:
                          type: string
                        prereleases:
                          properties:
                            identifiers:
                              items:
                                type: string
                              type: array
                          type: object
                      type: object
```

```
                        type: object
                      secretRef:
                        description: 'Secret with auth details. allowed keys: ssh-
privatekey,
                          ssh-knownhosts, username, password (optional) (if ssh-knownhosts
                          is not specified, git will not perform strict host checking)'
                        properties:
                          name:
                            description: Object is expected to be within same namespace
                            type: string
                        type: object
                      subPath:
                        description: Grab only portion of repository (optional)
                        type: string
                      url:
                        description: http or ssh urls are supported (required)
                        type: string
                    type: object
                  helmChart:
                    description: Uses helm fetch to fetch specified chart
                    properties:
                      name:
                        description: 'Example: stable/redis'
                        type: string
                      repository:
                        properties:
                          secretRef:
                            properties:
                              name:
                                description: Object is expected to be within same
                                  namespace
                                type: string
                            type: object
                          url:
                            description: Repository url; scheme of oci:// will fetch
                              experimental helm oci chart (v0.19.0+) (required)
                            type: string
                        type: object
                      version:
                        type: string
                    type: object
                  http:
                    description: Uses http library to fetch file
                    properties:
                      secretRef:
                        description: 'Secret to provide auth details (optional)
                          Secret may include one or more keys: username, password'
                        properties:
                          name:
                            description: Object is expected to be within same namespace
                            type: string
                        type: object
                      sha256:
                        description: Checksum to verify after download (optional)
                        type: string
```

```
                        subPath:
                          description: Grab only portion of download (optional)
                          type: string
                        url:
                          description: 'URL can point to one of following formats:
                            text, tgz, zip http and https url are supported; plain
                            file, tgz and tar types are supported (required)'
                          type: string
                      type: object
                    image:
                      description: Pulls content from Docker/OCI registry
                      properties:
                        secretRef:
                          description: 'Secret may include one or more keys: username,
                            password, token. By default anonymous access is used for
                            authentication.'
                          properties:
                            name:
                              description: Object is expected to be within same namespace
                              type: string
                          type: object
                        subPath:
                          description: Grab only portion of image (optional)
                          type: string
                        tagSelection:
                          description: Specifies a strategy to choose a tag (optional;
                            v0.24.0+) if specified, do not include a tag in url key
                          properties:
                            semver:
                              properties:
                                constraints:
                                  type: string
                                prereleases:
                                  properties:
                                    identifiers:
                                      items:
                                        type: string
                                      type: array
                                  type: object
                              type: object
                          type: object
                        url:
                          description: 'Docker image url; unqualified, tagged, or
                            digest references supported (required) Example: username/app1-
config:v0.1.0'
                          type: string
                      type: object
                    imgpkgBundle:
                      description: Pulls imgpkg bundle from Docker/OCI registry (v0.17.0+)
                      properties:
                        image:
                          description: Docker image url; unqualified, tagged, or digest
                            references supported (required)
                          type: string
                        secretRef:
```

```
            description: 'Secret may include one or more keys: username,
              password, token. By default anonymous access is used for
              authentication.'
            properties:
              name:
                description: Object is expected to be within same namespace
                type: string
            type: object
          tagSelection:
            description: Specifies a strategy to choose a tag (optional;
              v0.24.0+) if specified, do not include a tag in url key
            properties:
              semver:
                properties:
                  constraints:
                    type: string
                  prereleases:
                    properties:
                      identifiers:
                        items:
                          type: string
                        type: array
                    type: object
                type: object
            type: object
        type: object
      inline:
        description: Pulls content from within this resource; or other
          resources in the cluster
        properties:
          paths:
            additionalProperties:
              type: string
            description: Specifies mapping of paths to their content;
              not recommended for sensitive values as CR is not encrypted
              (optional)
            type: object
          pathsFrom:
            description: Specifies content via secrets and config maps;
              data values are recommended to be placed in secrets (optional)
            items:
              properties:
                configMapRef:
                  properties:
                    directoryPath:
                      description: Specifies where to place files found
                        in secret (optional)
                      type: string
                    name:
                      type: string
                  type: object
                secretRef:
                  properties:
                    directoryPath:
                      description: Specifies where to place files found
```

```
                              in secret (optional)
                            type: string
                          name:
                            type: string
                        type: object
                    type: object
                type: array
            type: object
          path:
            description: Relative path to place the fetched artifacts
            type: string
      type: object
    type: array
  noopDelete:
    description: Deletion requests for the App will result in the App
      CR being deleted, but its associated resources will not be deleted
      (optional; default=false; v0.18.0+)
    type: boolean
  paused:
    description: Pauses _future_ reconciliation; does _not_ affect currently
      running reconciliation (optional; default=false)
    type: boolean
  serviceAccountName:
    description: Specifies that app should be deployed authenticated via
      given service account, found in this namespace (optional; v0.6.0+)
    type: string
  syncPeriod:
    description: Specifies the length of time to wait, in time + unit
      format, before reconciling. Always >= 30s. If value below 30s is
      specified, 30s will be used. (optional; v0.9.0+; default=30s)
    type: string
  template:
    items:
      properties:
        cue:
          properties:
            inputExpression:
              description: Cue expression for single path component, can
                be used to unify ValuesFrom into a given field (optional)
              type: string
            outputExpression:
              description: Cue expression to output, default will export
                all visible fields (optional)
              type: string
            paths:
              description: Explicit list of files/directories (optional)
              items:
                type: string
              type: array
            valuesFrom:
              description: Provide values (optional)
              items:
                properties:
                  configMapRef:
                    properties:
```

```
      name:
        type: string
    type: object
  downwardAPI:
    properties:
      items:
        items:
          properties:
            fieldPath:
              description: 'Required: Selects a field
                of the app: only annotations, labels,
                uid, name and namespace are supported.'
              type: string
            kappControllerVersion:
              description: 'Optional: Get running KappController
                version, defaults (empty) to retrieving
                the current running version.. Can be manually
                supplied instead.'
              properties:
                version:
                  type: string
              type: object
            kubernetesAPIs:
              description: 'Optional: Get running KubernetesAPIs
                from cluster, defaults (empty) to retrieving
                the APIs from the cluster. Can be manually
                supplied instead, e.g ["group/version",
                "group2/version2"]'
              properties:
                groupVersions:
                  items:
                    type: string
                  type: array
              type: object
            kubernetesVersion:
              description: 'Optional: Get running Kubernetes
                version from cluster, defaults (empty)
                to retrieving the version from the cluster.
                Can be manually supplied instead.'
              properties:
                version:
                  type: string
              type: object
            name:
              type: string
          type: object
        type: array
    type: object
  path:
    type: string
  secretRef:
    properties:
      name:
        type: string
    type: object
```

```
            type: object
          type: array
      type: object
    helmTemplate:
      description: Use helm template command to render helm chart
      properties:
        kubernetesAPIs:
          description: 'Optional: Use kubernetes group/versions resources
            available in the live cluster'
          properties:
            groupVersions:
              items:
                type: string
              type: array
          type: object
        kubernetesVersion:
          description: 'Optional: Get Kubernetes version, defaults
            (empty) to retrieving the version from the cluster. Can
            be manually overridden to a value instead.'
          properties:
            version:
              type: string
          type: object
        name:
          description: Set name explicitly, default is App CR's name
            (optional; v0.13.0+)
          type: string
        namespace:
          description: Set namespace explicitly, default is App CR's
            namespace (optional; v0.13.0+)
          type: string
        path:
          description: Path to chart (optional; v0.13.0+)
          type: string
        valuesFrom:
          description: One or more secrets, config maps, paths that
            provide values (optional)
          items:
            properties:
              configMapRef:
                properties:
                  name:
                    type: string
                type: object
              downwardAPI:
                properties:
                  items:
                    items:
                      properties:
                        fieldPath:
                          description: 'Required: Selects a field
                            of the app: only annotations, labels,
                            uid, name and namespace are supported.'
                          type: string
                        kappControllerVersion:
```

```
                                    description: 'Optional: Get running KappController
                                      version, defaults (empty) to retrieving
                                      the current running version.. Can be manually
                                      supplied instead.'
                                    properties:
                                      version:
                                        type: string
                                    type: object
                                  kubernetesAPIs:
                                    description: 'Optional: Get running KubernetesAPIs
                                      from cluster, defaults (empty) to retrieving
                                      the APIs from the cluster. Can be manually
                                      supplied instead, e.g ["group/version",
                                      "group2/version2"]'
                                    properties:
                                      groupVersions:
                                        items:
                                          type: string
                                        type: array
                                    type: object
                                  kubernetesVersion:
                                    description: 'Optional: Get running Kubernetes
                                      version from cluster, defaults (empty)
                                      to retrieving the version from the cluster.
                                      Can be manually supplied instead.'
                                    properties:
                                      version:
                                        type: string
                                    type: object
                                  name:
                                    type: string
                                type: object
                              type: array
                            type: object
                          path:
                            type: string
                          secretRef:
                            properties:
                              name:
                                type: string
                            type: object
                        type: object
                    type: array
                type: object
              jsonnet:
                description: TODO implement jsonnet
                type: object
              kbld:
                description: Use kbld to resolve image references to use digests
                properties:
                  paths:
                    items:
                      type: string
                    type: array
                type: object
```

```
                    kustomize:
                      description: TODO implement kustomize
                      type: object
                    sops:
                      description: Use sops to decrypt *.sops.yml files (optional;
                        v0.11.0+)
                      properties:
                        age:
                          properties:
                            privateKeysSecretRef:
                              description: Secret with private armored PGP private
                                keys (required)
                              properties:
                                name:
                                  type: string
                              type: object
                          type: object
                        paths:
                          description: Lists paths to decrypt explicitly (optional;
                            v0.13.0+)
                          items:
                            type: string
                          type: array
                        pgp:
                          description: Use PGP to decrypt files (required)
                          properties:
                            privateKeysSecretRef:
                              description: Secret with private armored PGP private
                                keys (required)
                              properties:
                                name:
                                  type: string
                              type: object
                          type: object
                      type: object
                    ytt:
                      description: Use ytt to template configuration
                      properties:
                        fileMarks:
                          description: Control metadata about input files passed to
                            ytt (optional; v0.18.0+) see https://carvel.dev/ytt/docs/latest/
file-marks/
                            for more details
                          items:
                            type: string
                          type: array
                        ignoreUnknownComments:
                          description: Ignores comments that ytt doesn't recognize
                            (optional; default=false)
                          type: boolean
                        inline:
                          description: Specify additional files, including data values
                            (optional)
                          properties:
                            paths:
```

```
                              additionalProperties:
                                type: string
                              description: Specifies mapping of paths to their content;
                                not recommended for sensitive values as CR is not
                                encrypted (optional)
                              type: object
                            pathsFrom:
                              description: Specifies content via secrets and config
                                maps; data values are recommended to be placed in
                                secrets (optional)
                              items:
                                properties:
                                  configMapRef:
                                    properties:
                                      directoryPath:
                                        description: Specifies where to place files
                                          found in secret (optional)
                                        type: string
                                      name:
                                        type: string
                                    type: object
                                  secretRef:
                                    properties:
                                      directoryPath:
                                        description: Specifies where to place files
                                          found in secret (optional)
                                        type: string
                                      name:
                                        type: string
                                    type: object
                                type: object
                              type: array
                          type: object
                        paths:
                          description: Lists paths to provide to ytt explicitly (optional)
                          items:
                            type: string
                          type: array
                        strict:
                          description: Forces strict mode https://github.com/k14s/ytt/blob/
develop/docs/strict.md
                            (optional; default=false)
                          type: boolean
                        valuesFrom:
                          description: Provide values via ytt's --data-values-file
                            (optional; v0.19.0-alpha.9)
                          items:
                            properties:
                              configMapRef:
                                properties:
                                  name:
                                    type: string
                                type: object
                              downwardAPI:
                                properties:
```

```
                                items:
                                  items:
                                    properties:
                                      fieldPath:
                                        description: 'Required: Selects a field
                                          of the app: only annotations, labels,
                                          uid, name and namespace are supported.'
                                        type: string
                                      kappControllerVersion:
                                        description: 'Optional: Get running KappController
                                          version, defaults (empty) to retrieving
                                          the current running version.. Can be manually
                                          supplied instead.'
                                        properties:
                                          version:
                                            type: string
                                        type: object
                                      kubernetesAPIs:
                                        description: 'Optional: Get running KubernetesAPIs
                                          from cluster, defaults (empty) to retrieving
                                          the APIs from the cluster. Can be manually
                                          supplied instead, e.g ["group/version",
                                          "group2/version2"]'
                                        properties:
                                          groupVersions:
                                            items:
                                              type: string
                                            type: array
                                        type: object
                                      kubernetesVersion:
                                        description: 'Optional: Get running Kubernetes
                                          version from cluster, defaults (empty)
                                          to retrieving the version from the cluster.
                                          Can be manually supplied instead.'
                                        properties:
                                          version:
                                            type: string
                                        type: object
                                      name:
                                        type: string
                                    type: object
                                  type: array
                              type: object
                            path:
                              type: string
                            secretRef:
                              properties:
                                name:
                                  type: string
                              type: object
                          type: object
                        type: array
                    type: object
                type: object
            type: array
```

```
      type: object
    status:
      properties:
        conditions:
          items:
            properties:
              message:
                description: Human-readable message indicating details about
                  last transition.
                type: string
              reason:
                description: Unique, this should be a short, machine understandable
                  string that gives the reason for condition's last transition.
                  If it reports "ResizeStarted" that means the underlying persistent
                  volume is being resized.
                type: string
              status:
                type: string
              type:
                description: ConditionType represents reconciler state
                type: string
            required:
            - status
            - type
            type: object
          type: array
        consecutiveReconcileFailures:
          type: integer
        consecutiveReconcileSuccesses:
          type: integer
        deploy:
          properties:
            error:
              type: string
            exitCode:
              type: integer
            finished:
              type: boolean
            kapp:
              description: KappDeployStatus contains the associated AppCR deployed
                resources
              properties:
                associatedResources:
                  description: AssociatedResources contains the associated App
                    label, namespaces and GKs
                  properties:
                    groupKinds:
                      items:
                        description: GroupKind specifies a Group and a Kind,
                          but does not force a version.  This is useful for
                          identifying concepts during lookup stages without
                          having partially valid types
                        properties:
                          group:
                            type: string
```

```
                          kind:
                            type: string
                        required:
                        - group
                        - kind
                        type: object
                      type: array
                    label:
                      type: string
                    namespaces:
                      items:
                        type: string
                      type: array
                  type: object
            type: object
          startedAt:
            format: date-time
            type: string
          stderr:
            type: string
          stdout:
            type: string
          updatedAt:
            format: date-time
            type: string
        type: object
      fetch:
        properties:
          error:
            type: string
          exitCode:
            type: integer
          startedAt:
            format: date-time
            type: string
          stderr:
            type: string
          stdout:
            type: string
          updatedAt:
            format: date-time
            type: string
        type: object
      friendlyDescription:
        type: string
      inspect:
        properties:
          error:
            type: string
          exitCode:
            type: integer
          stderr:
            type: string
          stdout:
            type: string
```

```
                  updatedAt:
                    format: date-time
                    type: string
                type: object
              managedAppName:
                type: string
              observedGeneration:
                description: Populated based on metadata.generation when controller
                  observes a change to the resource; if this value is out of data,
                  other status fields do not reflect latest state
                format: int64
                type: integer
              template:
                properties:
                  error:
                    type: string
                  exitCode:
                    type: integer
                  stderr:
                    type: string
                  updatedAt:
                    format: date-time
                    type: string
                type: object
              usefulErrorMessage:
                type: string
            type: object
        required:
        - spec
        type: object
    served: true
    storage: true
    subresources:
      status: {}
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: packageinstalls.packaging.carvel.dev
spec:
  group: packaging.carvel.dev
  names:
    categories:
    - carvel
    kind: PackageInstall
    listKind: PackageInstallList
    plural: packageinstalls
    shortNames:
    - pkgi
    singular: packageinstall
  scope: Namespaced
  versions:
  - additionalPrinterColumns:
    - description: PackageMetadata name
      jsonPath: .spec.packageRef.refName
```

```
      name: Package name
      type: string
    - description: PackageMetadata version
      jsonPath: .status.version
      name: Package version
      type: string
    - description: Friendly description
      jsonPath: .status.friendlyDescription
      name: Description
      type: string
    - description: Time since creation
      jsonPath: .metadata.creationTimestamp
      name: Age
      type: date
    name: v1alpha1
    schema:
      openAPIV3Schema:
        description: A Package Install is an actual installation of a package and
          its underlying resources on a Kubernetes cluster. It is represented in kapp-
controller
          by a PackageInstall CR. A PackageInstall CR must reference a Package CR.
        properties:
          apiVersion:
            description: 'APIVersion defines the versioned schema of this representation
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources'
            type: string
          kind:
            description: 'Kind is a string value representing the REST resource this
              object represents. Servers may infer this from the endpoint the client
              submits requests to. Cannot be updated. In CamelCase. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds'
            type: string
          metadata:
            type: object
          spec:
            properties:
              canceled:
                description: Canceled when set to true will stop all active changes
                type: boolean
              cluster:
                description: Specifies that Package should be deployed to destination
                  cluster; by default, cluster is same as where this resource resides
                  (optional)
                properties:
                  kubeconfigSecretRef:
                    description: Specifies secret containing kubeconfig (required)
                    properties:
                      key:
                        description: Specifies key that contains kubeconfig (optional)
                        type: string
                      name:
                        description: Specifies secret name within app's namespace
                          (required)
```

```
                type: string
            type: object
          namespace:
            description: Specifies namespace in destination cluster (optional)
            type: string
        type: object
      noopDelete:
        description: When NoopDelete set to true, PackageInstall deletion
          should delete PackageInstall/App CR but preserve App's associated
          resources.
        type: boolean
      packageRef:
        description: Specifies the name of the package to install (required)
        properties:
          refName:
            type: string
          versionSelection:
            properties:
              constraints:
                type: string
              prereleases:
                properties:
                  identifiers:
                    items:
                      type: string
                    type: array
                type: object
            type: object
        type: object
      paused:
        description: Paused when set to true will ignore all pending changes,
          once it set back to false, pending changes will be applied
        type: boolean
      serviceAccountName:
        description: Specifies service account that will be used to install
          underlying package contents
        type: string
      syncPeriod:
        description: Controls frequency of App reconciliation in time + unit
          format. Always >= 30s. If value below 30s is specified, 30s will
          be used.
        type: string
      values:
        description: Values to be included in package's templating step (currently
          only included in the first templating step) (optional)
        items:
          properties:
            secretRef:
              properties:
                key:
                  type: string
                name:
                  type: string
              type: object
          type: object
```

```
                type: array
            type: object
          status:
            properties:
              conditions:
                items:
                  properties:
                    message:
                      description: Human-readable message indicating details about
                        last transition.
                      type: string
                    reason:
                      description: Unique, this should be a short, machine understandable
                        string that gives the reason for condition's last transition.
                        If it reports "ResizeStarted" that means the underlying persistent
                        volume is being resized.
                      type: string
                    status:
                      type: string
                    type:
                      description: ConditionType represents reconciler state
                      type: string
                  required:
                  - status
                  - type
                  type: object
                type: array
              friendlyDescription:
                type: string
              lastAttemptedVersion:
                description: LastAttemptedVersion specifies what version was last
                  attempted to be installed. It does _not_ indicate it was successfully
                  installed.
                type: string
              observedGeneration:
                description: Populated based on metadata.generation when controller
                  observes a change to the resource; if this value is out of data,
                  other status fields do not reflect latest state
                format: int64
                type: integer
              usefulErrorMessage:
                type: string
              version:
                description: TODO this is desired resolved version (not actually deployed)
                type: string
            type: object
        required:
        - spec
        type: object
    served: true
    storage: true
    subresources:
      status: {}
---
apiVersion: apiextensions.k8s.io/v1
```

```
kind: CustomResourceDefinition
metadata:
  annotations:
    packaging.carvel.dev/global-namespace: kapp-controller-packaging-global
  name: packagerepositories.packaging.carvel.dev
spec:
  group: packaging.carvel.dev
  names:
    categories:
    - carvel
    kind: PackageRepository
    listKind: PackageRepositoryList
    plural: packagerepositories
    shortNames:
    - pkgr
    singular: packagerepository
  scope: Namespaced
  versions:
  - additionalPrinterColumns:
    - description: Time since creation
      jsonPath: .metadata.creationTimestamp
      name: Age
      type: date
    - description: Friendly description
      jsonPath: .status.friendlyDescription
      name: Description
      type: string
    name: v1alpha1
    schema:
      openAPIV3Schema:
        description: A package repository is a collection of packages and their metadata.
          Similar to a maven repository or a rpm repository, adding a package repository
          to a cluster gives users of that cluster the ability to install any of the
          packages from that repository.
        properties:
          apiVersion:
            description: 'APIVersion defines the versioned schema of this representation
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources'
            type: string
          kind:
            description: 'Kind is a string value representing the REST resource this
              object represents. Servers may infer this from the endpoint the client
              submits requests to. Cannot be updated. In CamelCase. More info: https://
git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds'
            type: string
          metadata:
            type: object
          spec:
            properties:
              fetch:
                properties:
                  git:
                    description: Uses git to clone repository containing package list
```

```
      properties:
        lfsSkipSmudge:
          description: Skip lfs download (optional)
          type: boolean
        ref:
          description: Branch, tag, commit; origin is the name of the
            remote (optional)
          type: string
        refSelection:
          description: Specifies a strategy to resolve to an explicit
            ref (optional; v0.24.0+)
          properties:
            semver:
              properties:
                constraints:
                  type: string
                prereleases:
                  properties:
                    identifiers:
                      items:
                        type: string
                      type: array
                  type: object
              type: object
          type: object
        secretRef:
          description: 'Secret with auth details. allowed keys: ssh-privatekey,
            ssh-knownhosts, username, password (optional) (if ssh-knownhosts
            is not specified, git will not perform strict host checking)'
          properties:
            name:
              description: Object is expected to be within same namespace
              type: string
          type: object
        subPath:
          description: Grab only portion of repository (optional)
          type: string
        url:
          description: http or ssh urls are supported (required)
          type: string
      type: object
    http:
      description: Uses http library to fetch file containing packages
      properties:
        secretRef:
          description: 'Secret to provide auth details (optional) Secret
            may include one or more keys: username, password'
          properties:
            name:
              description: Object is expected to be within same namespace
              type: string
          type: object
        sha256:
          description: Checksum to verify after download (optional)
          type: string
```

```
                    subPath:
                      description: Grab only portion of download (optional)
                      type: string
                    url:
                      description: 'URL can point to one of following formats: text,
                        tgz, zip http and https url are supported; plain file, tgz
                        and tar types are supported (required)'
                      type: string
                  type: object
                image:
                  description: Image url; unqualified, tagged, or digest references
                    supported (required)
                  properties:
                    secretRef:
                      description: 'Secret may include one or more keys: username,
                        password, token. By default anonymous access is used for
                        authentication.'
                      properties:
                        name:
                          description: Object is expected to be within same namespace
                          type: string
                      type: object
                    subPath:
                      description: Grab only portion of image (optional)
                      type: string
                    tagSelection:
                      description: Specifies a strategy to choose a tag (optional;
                        v0.24.0+) if specified, do not include a tag in url key
                      properties:
                        semver:
                          properties:
                            constraints:
                              type: string
                            prereleases:
                              properties:
                                identifiers:
                                  items:
                                    type: string
                                  type: array
                              type: object
                          type: object
                      type: object
                    url:
                      description: 'Docker image url; unqualified, tagged, or digest
                        references supported (required) Example: username/app1-
config:v0.1.0'
                      type: string
                  type: object
                imgpkgBundle:
                  description: Pulls imgpkg bundle from Docker/OCI registry
                  properties:
                    image:
                      description: Docker image url; unqualified, tagged, or digest
                        references supported (required)
                      type: string
```

```
      secretRef:
        description: 'Secret may include one or more keys: username,
          password, token. By default anonymous access is used for
          authentication.'
        properties:
          name:
            description: Object is expected to be within same namespace
            type: string
        type: object
      tagSelection:
        description: Specifies a strategy to choose a tag (optional;
          v0.24.0+) if specified, do not include a tag in url key
        properties:
          semver:
            properties:
              constraints:
                type: string
              prereleases:
                properties:
                  identifiers:
                    items:
                      type: string
                    type: array
                type: object
            type: object
        type: object
    type: object
  inline:
    description: Pull content from within this resource; or other
      resources in the cluster
    properties:
      paths:
        additionalProperties:
          type: string
        description: Specifies mapping of paths to their content;
          not recommended for sensitive values as CR is not encrypted
          (optional)
        type: object
      pathsFrom:
        description: Specifies content via secrets and config maps;
          data values are recommended to be placed in secrets (optional)
        items:
          properties:
            configMapRef:
              properties:
                directoryPath:
                  description: Specifies where to place files found
                    in secret (optional)
                  type: string
                name:
                  type: string
              type: object
            secretRef:
              properties:
                directoryPath:
```

```
                              description: Specifies where to place files found
                                in secret (optional)
                              type: string
                            name:
                              type: string
                          type: object
                        type: object
                      type: array
                  type: object
              type: object
            paused:
              description: Paused when set to true will ignore all pending changes,
                once it set back to false, pending changes will be applied
              type: boolean
            syncPeriod:
              description: Controls frequency of PackageRepository reconciliation
              type: string
          required:
          - fetch
          type: object
        status:
          properties:
            conditions:
              items:
                properties:
                  message:
                    description: Human-readable message indicating details about
                      last transition.
                    type: string
                  reason:
                    description: Unique, this should be a short, machine understandable
                      string that gives the reason for condition's last transition.
                      If it reports "ResizeStarted" that means the underlying persistent
                      volume is being resized.
                    type: string
                  status:
                    type: string
                  type:
                    description: ConditionType represents reconciler state
                    type: string
                required:
                - status
                - type
                type: object
              type: array
            consecutiveReconcileFailures:
              type: integer
            consecutiveReconcileSuccesses:
              type: integer
            deploy:
              properties:
                error:
                  type: string
                exitCode:
                  type: integer
```

```
                finished:
                  type: boolean
                kapp:
                  description: KappDeployStatus contains the associated AppCR deployed
                    resources
                  properties:
                    associatedResources:
                      description: AssociatedResources contains the associated App
                        label, namespaces and GKs
                      properties:
                        groupKinds:
                          items:
                            description: GroupKind specifies a Group and a Kind,
                              but does not force a version.  This is useful for
                              identifying concepts during lookup stages without
                              having partially valid types
                            properties:
                              group:
                                type: string
                              kind:
                                type: string
                            required:
                            - group
                            - kind
                            type: object
                          type: array
                        label:
                          type: string
                        namespaces:
                          items:
                            type: string
                          type: array
                      type: object
                  type: object
                startedAt:
                  format: date-time
                  type: string
                stderr:
                  type: string
                stdout:
                  type: string
                updatedAt:
                  format: date-time
                  type: string
              type: object
            fetch:
              properties:
                error:
                  type: string
                exitCode:
                  type: integer
                startedAt:
                  format: date-time
                  type: string
                stderr:
```

```
                    type: string
                  stdout:
                    type: string
                  updatedAt:
                    format: date-time
                    type: string
                type: object
              friendlyDescription:
                type: string
              observedGeneration:
                description: Populated based on metadata.generation when controller
                  observes a change to the resource; if this value is out of data,
                  other status fields do not reflect latest state
                format: int64
                type: integer
              template:
                properties:
                  error:
                    type: string
                  exitCode:
                    type: integer
                  stderr:
                    type: string
                  updatedAt:
                    format: date-time
                    type: string
                type: object
              usefulErrorMessage:
                type: string
            type: object
        required:
        - spec
        type: object
    served: true
    storage: true
    subresources:
      status: {}
---
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    kapp-controller.carvel.dev/version: v0.45.2
    kbld.k14s.io/images: |
      - origins:
        - local:
            path: /home/runner/work/kapp-controller/kapp-controller
        - git:
            dirty: true
            remoteURL: https://github.com/carvel-dev/kapp-controller
            sha: e3beee23d49899bfc681c9d980c1a3bdc0fa14ac
            tags:
            - v0.45.2
        url: ghcr.io/carvel-dev/kapp-
controller@sha256:d5c5b259d10f8a561fe6717a735ceb053ccb13320f55428977d1d8df46b9bc0d
```

```
  name: kapp-controller
  namespace: tkg-system
spec:
  replicas: 1
  revisionHistoryLimit: 0
  selector:
    matchLabels:
      app: kapp-controller
  template:
    metadata:
      labels:
        app: kapp-controller
    spec:
      containers:
      - args:
        - -packaging-global-namespace=kapp-controller-packaging-global
        - -enable-api-priority-and-fairness=True
        - -tls-cipher-suites=
        env:
        - name: KAPPCTRL_MEM_TMP_DIR
          value: /etc/kappctrl-mem-tmp
        - name: KAPPCTRL_SIDECAREXEC_SOCK
          value: /etc/kappctrl-mem-tmp/sidecarexec.sock
        - name: KAPPCTRL_SYSTEM_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: KAPPCTRL_API_PORT
          value: "10350"
        image: ghcr.io/carvel-dev/kapp-
controller@sha256:d5c5b259d10f8a561fe6717a735ceb053ccb13320f55428977d1d8df46b9bc0d
        name: kapp-controller
        ports:
        - containerPort: 10350
          name: api
          protocol: TCP
        resources:
          requests:
            cpu: 120m
            memory: 100Mi
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          readOnlyRootFilesystem: true
          runAsNonRoot: true
          seccompProfile:
            type: RuntimeDefault
        volumeMounts:
        - mountPath: /etc/kappctrl-mem-tmp
          name: template-fs
        - mountPath: /home/kapp-controller
          name: home
      - args:
```

```
        - --sidecarexec
        env:
        - name: KAPPCTRL_SIDECAREXEC_SOCK
          value: /etc/kappctrl-mem-tmp/sidecarexec.sock
        - name: IMGPKG_ACTIVE_KEYCHAINS
          value: gke,aks,ecr
        image: ghcr.io/carvel-dev/kapp-
controller@sha256:d5c5b259d10f8a561fe6717a735ceb053ccb13320f55428977d1d8df46b9bc0d
        name: kapp-controller-sidecarexec
        resources:
          requests:
            cpu: 120m
            memory: 100Mi
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          readOnlyRootFilesystem: false
          runAsNonRoot: true
          seccompProfile:
            type: RuntimeDefault
        volumeMounts:
        - mountPath: /etc/kappctrl-mem-tmp
          name: template-fs
        - mountPath: /home/kapp-controller
          name: home
        - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
          name: empty-sa
      serviceAccount: kapp-controller-sa
      volumes:
      - emptyDir:
          medium: Memory
        name: template-fs
      - emptyDir:
          medium: Memory
        name: home
      - emptyDir: {}
        name: empty-sa
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kapp-controller-sa
  namespace: tkg-system
```

Installieren von Cert Manager auf TKr für vSphere 7.x

Lesen Sie diese Anweisungen, um Cert Manager auf einem TKr für vSphere 7.x zu installieren.

Voraussetzungen

Weitere Informationen hierzu finden Sie unter Workflow zum Installieren von Standardpaketen auf TKr für vSphere 7.x.

Installieren des Zertifikatmanagers

Installieren Sie Cert Manager.

1 Listet die verfügbaren Versionen des Cert Manager-Pakets auf.

```
kubectl -n tkg-system get packages | grep cert-manager
```

2 Erstellen Sie `cert-manager.yaml` mit der Zielversion.

Weitere Informationen hierzu finden Sie unter cert-manager.yaml.

3 Installieren Sie Cert Manager.

```
kubectl apply -f cert-manager.yaml
```

Erwartetes Ergebnis:

```
serviceaccount/cert-manager-sa created
clusterrolebinding.rbac.authorization.k8s.io/admin created
packageinstall.packaging.carvel.dev/cert-manager created
secret/cert-manager-data-values created
```

4 Überprüfen Sie die Cert Manager-Installation.

```
kubectl get pkgi -A
```

Erwartetes Ergebnis:

```
NAMESPACE    NAME           PACKAGE NAME                    PACKAGE VERSION
DESCRIPTION           AGE
tkg-system   cert-manager   cert-manager.tanzu.vmware.com   1.12.2+vmware.2-
tkg.2   Reconcile succeeded   57s
```

5 Überprüfen Sie die Cert Manager-Pods.

```
kubectl get pods -A
```

```
NAMESPACE           NAME                                       READY   STATUS
RESTARTS      AGE
tkg-system        cert-manager-666586c866-826rz               1/1     Running
0             48s
tkg-system        cert-manager-cainjector-68697ccc4b-xbfff    1/1     Running
0             48s
tkg-system        cert-manager-webhook-57ccbd4db9-tzw4c       1/1     Running
0             48s
```

cert-manager.yaml

Weitere Informationen zur Cert Manager-Installation finden Sie im folgenden `cert-manager.yaml`-Beispiel. Aktualisieren Sie die Versionsvariable so, dass sie mit der Zielpaketversion übereinstimmt.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cert-manager-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: cert-manager-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: cert-manager
  namespace: tkg-system
spec:
  serviceAccountName: cert-manager-sa
  packageRef:
    refName: cert-manager.tanzu.vmware.com
    versionSelection:
      constraints: 1.12.2+vmware.2-tkg.2 #PKG-VERSION
  values:
  - secretRef:
      name: cert-manager-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: cert-manager-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: tkg-system
```

Installieren von Contour auf TKr für vSphere 7.x

Lesen Sie diese Anweisungen zum Installieren von Standardpaketen auf einem TKG-Cluster, der mit einer TKr für vSphere 7.x bereitgestellt wird.

Voraussetzungen

Weitere Informationen hierzu finden Sie unter Workflow zum Installieren von Standardpaketen auf TKr für vSphere 7.x.

Installieren von Contour mit Envoy

Installieren Sie Contour Ingress mit dem Envoy-Dienst.

1 Listet die verfügbaren Contour-Versionen im Repository auf.

```
kubectl get packages -n tkg-system | grep contour
```

2 Erstellen Sie eine `contour.yaml`-Spezifikation.

Weitere Informationen hierzu finden Sie unter #unique_187/unique_187_Connect_42_GUID-CC995CF8-0F4B-4D92-A782-A3832C0EA5AE.

3 Passen Sie bei Bedarf `contour-data-values` für Ihre Umgebung an.

Weitere Informationen hierzu finden Sie unter Referenz zum Contour-Paket.

4 Installieren Sie Contour.

```
kubectl apply -f contour.yaml
```

```
serviceaccount/contour-sa
createdclusterrolebinding.rbac.authorization.k8s.io/contour-role-binding created
packageinstall.packaging.carvel.dev/contour created
secret/contour-data-values created
```

5 Überprüfen Sie die Contour-Paketinstallation.

```
kubectl get pkgi -A
```

6 Überprüfen Sie die Contour-Objekte.

```
kubectl get all -n contour-ingress
```

```
NAME                            READY   STATUS    RESTARTS   AGE
pod/contour-777bdddc69-fqnsp    1/1     Running   0          102s
pod/contour-777bdddc69-gs5xv    1/1     Running   0          102s
pod/envoy-d4jtt                 2/2     Running   0          102s
pod/envoy-g5h72                 2/2     Running   0          102s
pod/envoy-pjpzc                 2/2     Running   0          102s

NAME              TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)                      AGE
service/contour   ClusterIP      10.105.242.46   <none>
8001/TCP                     102s
service/envoy     LoadBalancer   10.103.245.57   10.197.154.69   80:32642/
TCP,443:30297/TCP   102s

NAME                   DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
```

```
SELECTOR   AGE
daemonset.apps/envoy   3         3         3       3            3
<none>          102s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/contour    2/2     2            2           102s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/contour-777bdddc69    2         2         2       102s
```

Das Contour-Paket installiert 2 Contour-Pods und 3 Envoy-Pods. Sowohl Contour als auch Envoy werden als Dienste bereitgestellt. In diesem Beispiel hat der Envoy-Dienst die externe IP-Adresse 10.197.154.69. Diese IP-Adresse wird vom CIDR getrennt, das für **Arbeitslastnetzwerk > Ingress** angegeben ist. Eine Lastausgleichsdienst-Instanz wird für diese IP-Adresse erstellt. Die Mitglieder des Serverpools für diesen Lastausgleichsdienst sind die Envoy-Pods. Die Envoy-Pods übernehmen die IP-Adressen der Worker-Knoten, auf denen sie ausgeführt werden. Sie können diese IPs anzeigen, indem Sie die Clusterknoten (`kubectl get nodes -o wide`) abfragen.

contour.yaml

Verwenden Sie die folgende `contour.yaml`, um Contour mit Envoy zu installieren. Aktualisieren Sie die Versionsvariable so, dass sie mit der Zielpaketversion übereinstimmt.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: contour-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: contour-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: contour-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: contour
  namespace: tkg-system
spec:
  serviceAccountName: contour-sa
  packageRef:
    refName: contour.tanzu.vmware.com
    versionSelection:
```

```
      constraints: 1.26.1+vmware.1-tkg.1 #PKG-VERSION
  values:
  - secretRef:
      name: contour-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: contour-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: tanzu-system-ingress
    contour:
      configFileContents: {}
      useProxyProtocol: false
      replicas: 2
      pspNames: "vmware-system-restricted"
      logLevel: info
    envoy:
      service:
        type: LoadBalancer
        annotations: {}
        externalTrafficPolicy: Cluster
        disableWait: false
      hostPorts:
        enable: true
        http: 80
        https: 443
      hostNetwork: false
      terminationGracePeriodSeconds: 300
      logLevel: info
    certificates:
      duration: 8760h
      renewBefore: 360h
```

Installieren von ExternalDNS auf TKr für vSphere 7.x

Lesen Sie diese Anweisungen zum Installieren von ExternalDNS auf einem TKG-Cluster, der mit einer TKr für vSphere 7.x bereitgestellt wurde.

Voraussetzungen

Weitere Informationen hierzu finden Sie unter Workflow zum Installieren von Standardpaketen auf TKr für vSphere 7.x.

Installieren von ExternalDNS

Installieren Sie ExternalDNS auf einem TKG-Cluster, der mit TKr für vSphere 7.x bereitgestellt wurde.

1 Listen Sie die verfügbaren ExternalDNS-Versionen im Repository auf.

```
kubectl get packages -n tkg-system | grep external-dns
```

2 Erstellen Sie den ExternalDNS-Namespace.

```
kubectl create namespace tanzu-system-service-discovery --dry-run=client -o yaml | kubectl
apply -f -
```

3 Legen Sie die Sicherheitsposition im Namespace fest.

```
kubectl label namespace tanzu-system-service-discovery pod-security.kubernetes.io/
enforce=privileged
```

4 Bereiten Sie die YAML-Datei für die Bind-Bereitstellung vor.

Weitere Informationen hierzu finden Sie unter bind-deployment.yaml.

5 Stellen Sie den BIND DNS-Server bereit.

```
kubectl apply -n tanzu-system-service-discovery -f bind-deployment.yaml
```

6 Bereiten Sie die YAML-Datei für die ExternalDNS-Bereitstellung vor.

Weitere Informationen hierzu finden Sie unter external-dns-deploy.yaml.

7 Erstellen Sie einen geheimen Schlüssel mithilfe der Datei `external-dns-default-values.yaml`.

```
svcip=$(kubectl get svc bind -n tanzu-system-service-discovery -o
jsonpath='{.spec.clusterIP}')sed -i "s/--rfc2136-host=[0-9.]\+/--rfc2136-host=$svcip/g"
external-dns-deploy.yaml
```

```
kubectl create secret generic external-dns-default-values --from-file=values.yaml=external-
dns-deploy.yaml -n tkg-system
```

8 Überprüfen Sie den geheimen Schlüssel.

```
kubectl get secret external-dns-default-values -n tkg-system
```

```
kubectl get secret external-dns-default-values -n tkg-system -oyaml
```

9 Bereiten Sie die YAML-Installationsdatei für das ExternalDNS-Paket vor.

Weitere Informationen hierzu finden Sie unter external-dns-packageinstall.yaml.

10 Konfigurieren Sie Bind.

```
sed -i "s/--rfc2136-host=[0-9.]\+/--rfc2136-host=$svcip/g" external-dns-packageinstall.yaml
```

11 Erstellen Sie das ExternalDNS-Paket.

```
kubectl apply -f external-dns-packageinstall.yaml
```

12 Überprüfen Sie die ExternalDNS-Installation.

```
kubectl get all -n tanzu-system-service-discovery
```

bind-deployment.yaml

Beispiel: `bind-deployment.yaml`.

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: bind-config
data:
  named.conf: |
    key "externaldns-key" {
            algorithm hmac-sha256;
            secret "O0DhTJzZ0GjfuQmB9TBc1ELchv5oDMTlQs3NNOdMZJU=";
    };

    # bind needs to recurse to coredns in the case of resolving CNAME records
    # it may know about to A records. E.g This test runs on AWS which uses
    # CNAMEs for their LoadBalancer Services and bind will want to resolve
    # those CNAME records to A records using an upstream DNS server.
    options {
        recursion yes;
        forwarders {
            COREDNS_CLUSTER_IP;
        };
        forward only;
        dnssec-enable yes;
        dnssec-validation yes;
    };

    zone "k8s.example.org" {
        type master;
        file "/etc/bind/k8s.zone";
        allow-transfer {
            key "externaldns-key";
        };
        update-policy {
            grant externaldns-key zonesub ANY;
        };
    };
  k8s.zone: |
    $TTL 60 ; 1 minute
    @         IN SOA  k8s.example.org. root.k8s.example.org. (
                                  16         ; serial
                                  60         ; refresh (1 minute)
                                  60         ; retry (1 minute)
```

```
                                    60         ; expire (1 minute)
                                    60         ; minimum (1 minute)
                                    )
                            NS      ns.k8s.example.org.
    ns                      A       1.2.3.4
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bind
spec:
  selector:
    matchLabels:
      app: bind
  template:
    metadata:
      labels:
        app: bind
    spec:
      containers:
      - name: bind
        image: docker.io/internetsystemsconsortium/bind9:9.16
        imagePullPolicy: IfNotPresent
        command:
        - 'sh'
        - '-c'
        - |
          /usr/sbin/named -g -c /etc/bind/named.conf
        ports:
        - containerPort: 53
          name: dns
          protocol: UDP
        - containerPort: 53
          name: dns-tcp
          protocol: TCP
        volumeMounts:
          - name: named-conf-volume
            mountPath: /etc/bind/named.conf
            subPath: named.conf
          - name: k8s-zone-volume
            mountPath: /etc/bind/k8s.zone
            subPath: k8s.zone
      volumes:
        - name: data
          emptyDir: {}
        - name: named-conf-volume
          configMap:
            name: bind-config
            items:
            - key: named.conf
              path: named.conf
        - name: k8s-zone-volume
          configMap:
            name: bind-config
            items:
```

```
            - key: k8s.zone
              path: k8s.zone
---
apiVersion: v1
kind: Service
metadata:
  name: bind
  labels:
    app: bind
spec:
  selector:
    app: bind
  type: ClusterIP
  ports:
  - port: 53
    targetPort: 53
    protocol: TCP
    name: dns-tcp
  - port: 53
    targetPort: 53
    protocol: UDP
    name: dns
```

external-dns-deploy.yaml

Beispiel: `external-dns-deploy.yaml`.

```
deployment:
  args:
  - --source=service
  - --source=ingress
  - --txt-owner-id=k8s
  - --domain-filter=k8s.example.org
  - --namespace=default
  - --provider=rfc2136
  - --rfc2136-host=198.201.49.227
  - --rfc2136-port=53
  - --rfc2136-zone=k8s.example.org
  - --rfc2136-tsig-secret=O0DhTJzZ0GjfuQmB9TBc1ELchv5oDMTlQs3NNOdMZJU=
  - --rfc2136-tsig-secret-alg=hmac-sha256
  - --rfc2136-tsig-keyname=externaldns-key
```

external-dns-packageinstall.yaml

Das folgende Beispiel kann für BIND verwendet werden. Aktualisieren Sie die Paketversion nach Bedarf.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: external-dns-default-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: ClusterRoleBinding
metadata:
  name: dns-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: external-dns-default-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: dns
  namespace: tkg-system
spec:
  serviceAccountName: external-dns-default-sa
  packageRef:
    refName: external-dns.tanzu.vmware.com
    versionSelection:
      constraints: 0.13.6+vmware.1-tkg.1
  values:
  - secretRef:
      name: external-dns-default-values
---
apiVersion: v1
kind: Secret
metadata:
  name: external-dns-reg-creds
  namespace: tanzu-system-service-discovery
stringData:
  values.yml: |
    ---
    namespace: tanzu-system-service-discovery
    dns:
      deployment:
        args:
        - --txt-owner-id=k8s
        - --provider=rfc2136
        - --rfc2136-host=198.201.49.227 #! IP of compatible DNS server
        - --rfc2136-port=53
        - --rfc2136-zone=mk8s.example.org #! zone where services are deployed
        - --rfc2136-tsig-secret=O0DhTJzZ0GjfuQmB9TBc1ELchv5oDMTlQs3NNOdMZJU= #! TSIG secret
authorized to update DNS
        - --rfc2136-tsig-secret-alg=hmac-sha256
        - --rfc2136-tsig-keyname=externaldns-key
        - --rfc2136-tsig-axfr
        - --source=service
        - --source=ingress
        - --domain-filter=k8s.example.org1 #! zone where services are deployed
```

Das folgende Beispiel kann für einen AWS DNS-Anbieter (Route 53) verwendet werden. Aktualisieren Sie die Paketversion nach Bedarf.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dns-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dns-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: dns-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: dns
  namespace: tkg-system
spec:
  serviceAccountName: dns-sa
  packageRef:
    refName: dns.tanzu.vmware.com
    versionSelection:
      constraints: 0.13.6+vmware.1-tkg.1
  values:
  - secretRef:
      name: dns-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: dns-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: tanzu-system-service-discovery
    dns:
      pspNames: "vmware-system-restricted"
      deployment:
        args:
        - --source=service
        - --source=ingress
        - --source=contour-httpproxy #! configure external-dns to read Contour HTTPProxy
resources
        - --domain-filter=my-zone.example.org #! zone where services are deployed
```

```
        - --provider=aws
        - --policy=upsert-only #! prevent deleting any records, omit to enable full sync
        - --aws-zone-type=public #! only look at public hosted zones (public, private, no
value for both)
        - --aws-prefer-cname
        - --registry=txt
        - --txt-owner-id=ROUTE_53_HOSTED_ZONE_ID #! Route53 hosted zone identifier for my-
zone.example.org
        - --txt-prefix=txt #! disambiguates TXT records from CNAME records
        env:
          - name: AWS_ACCESS_KEY_ID
            valueFrom:
              secretKeyRef:
                name: route53-credentials #! Kubernetes secret for route53 credentials
                key: aws_access_key_id
          - name: AWS_SECRET_ACCESS_KEY
            valueFrom:
              secretKeyRef:
                name: route53-credentials #! Kubernetes secret for route53 credentials
                key: aws_secret_access_key
```

Das folgende Beispiel kann für einen Azure DNS-Anbieter verwendet werden. Aktualisieren Sie die Paketversion nach Bedarf.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dns-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dns-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: dns-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: dns
  namespace: tkg-system
spec:
  serviceAccountName: dns-sa
  packageRef:
    refName: dns.tanzu.vmware.com
    versionSelection:
      constraints: 0.13.6+vmware.1-tkg.1
  values:
```

```
  - secretRef:
      name: dns-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: dns-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: tanzu-system-service-discovery
    dns:
      pspNames: "vmware-system-restricted"
      deployment:
        args:
        - --provider=azure
        - --source=service
        - --source=ingress
        - --source=contour-httpproxy #! read Contour HTTPProxy resources
        - --domain-filter=my-zone.example.org #! zone where services are deployed
        - --azure-resource-group=my-resource-group #! Azure resource group
        volumeMounts:
        - name: azure-config-file
          mountPath: /etc/kubernetes
          readOnly: true
        #@overlay/replace
        volumes:
        - name: azure-config-file
          secret:
            secretName: azure-config-file
```

Installieren von Fluent Bit auf TKr für vSphere 7.x

Lesen Sie diese Anweisungen zum Installieren von Fluent Bit auf einem TKG-Cluster, der mit TKr für vSphere 7.x bereitgestellt wurde.

Voraussetzungen

Weitere Informationen hierzu finden Sie unter Workflow zum Installieren von Standardpaketen auf TKr für vSphere 7.x.

Installieren von Fluent Bit

Installieren Sie Fluent Bit für die Protokollweiterleitung.

1 Listet die verfügbaren Fluent Bit-Versionen im Repository auf.

```
kubectl -n tkg-system get packages | grep fluent-bit
```

2 Erstellen Sie den Namespace.

```
kubectl create ns tanzu-system-logging
```

3 Beschriften Sie den Namespace für PSA.

```
kubectl label ns fluentbit-logging pod-security.kubernetes.io/enforce=privileged
```

Alternativ:

```
apiVersion: v1
kind: Namespace
metadata:
  name: fluentbit-logging
---
apiVersion: v1
kind: Namespace
metadata:
  name: fluentbit-logging
  labels: pod-security.kubernetes.io/enforce: privileged
```

4 Bereiten Sie die Datei `fluentbit.yaml` vor.

Informationen hierzu finden Sie unter

5 Passen Sie `fluentbit-data-values` nach Bedarf für Ihre Umgebung an.

Informationen zu Konfigurationsoptionen finden Sie unter Referenz zum Fluent Bit-Paket.

6 Installieren Sie Fluent Bit.

```
kubectl apply -f fluentbit.yaml
```

7 Überprüfen Sie die Fluent Bit-Installation.

```
kubectl get all -n fluentbit-logging
```

fluentbit.yaml

Das folgende Beispiel kann für einen Syslog-Endpoint verwendet werden. Aktualisieren Sie die Paketversion nach Bedarf.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: fluentbit-sa
  namespace: tkg-system
  annotations:
    pod-security.kubernetes.io/enforce: "privileged"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: fluentbit-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
```

```
subjects:
  - kind: ServiceAccount
    name: fluentbit-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: fluentbit
  namespace: tkg-system
spec:
  serviceAccountName: fluentbit-sa
  packageRef:
    refName: fluent-bit.tanzu.vmware.com
    versionSelection:
      constraints: 2.1.6+vmware.1-tkg.2 #PKG_VERSION
  values:
  - secretRef:
      name: fluentbit-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: fluentbit-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: tanzu-system-logging
    tkg:
      instance_name: "guest-cluster"     #TKG_INSTANCE_NAME
      cluster_name: "tkgs-vc-wl"         #TKG_CLUSTER_NAME
    fluentbit:
      output_plugin: "syslog"
      syslog:
        host: "10.202.27.235"           #SYSLOG_HOST
        port: "514"                     #SYSLOG_PORT
        mode: "tcp"                     #SYSLOG_MODE
        format: "rfc5424"               #SYSLOG_FORMAT
```

Installieren von Prometheus auf TKr für vSphere 7.x

Lesen Sie diese Anweisungen zum Installieren von Prometheus auf einem TKG-Cluster, der mit TKr für vSphere 7.x bereitgestellt wurde.

Voraussetzungen

Weitere Informationen hierzu finden Sie unter Workflow zum Installieren von Standardpaketen auf TKr für vSphere 7.x.

Installieren von Prometheus

Installieren Sie Prometheus mit Alertmanager.

1 Listen Sie die verfügbaren Prometheus-Paketversionen im Repository auf.

```
kubectl get packages -n tkg-system | grep prometheus
```

2 Erstellen Sie den Prometheus-Namespace.

```
kubectl create ns tanzu-system-monitoring
```

3 Konfigurieren Sie PSA im Prometheus-Namespace.

```
kubectl label ns prometheus-monitoring pod-security.kubernetes.io/enforce=privileged
```

```
kubectl get ns prometheus-monitoring -oyaml|grep privileged
```

4 Erstellen Sie die `prometheus-data-values.yaml`-Datei.

Weitere Informationen hierzu finden Sie unter .

5 Erstellen Sie einen geheimen Schlüssel mithilfe von `prometheus-data-values.yaml` als Eingabe.

Hinweis Aufgrund der Größe des geheimen `prometheus-data-values`-Schlüssels ist es weniger fehleranfällig, wenn Sie ihn separat erstellen und nicht versuchen, ihn in die Prometheus-YAML-Spezifikation aufzunehmen.

```
kubectl create secret generic prometheus-data-values --from-file=values.yaml=prometheus-data-values.yaml -n tkg-system
```

```
secret/prometheus-data-values created
```

6 Überprüfen Sie den geheimen Schlüssel.

```
kubectl get secrets -A
```

```
kubectl describe secret prometheus-data-values -n tkg-system
```

7 Passen Sie bei Bedarf die `prometheus-data-values` für Ihre Umgebung an.

Weitere Informationen hierzu finden Sie unter Prometheus-Konfiguration.

Wenn Sie `prometheus-data-values.yaml` aktualisieren, ersetzen Sie den geheimen Schlüssel mit diesem Befehl.

```
kubectl create secret generic prometheus-data-values --from-file=values.yaml=prometheus-data-values.yaml -n tkg-system -o yaml --dry-run=client | kubectl replace -f-
```

```
secret/prometheus-data-values replaced
```

8 Erstellen Sie eine `prometheus.yaml`-Spezifikation.

Weitere Informationen hierzu finden Sie unter Installieren von Prometheus auf TKr für vSphere 7.x.

9 Installieren Sie Prometheus.

```
kubectl apply -f prometheus.yaml
```

```
serviceaccount/prometheus-sa created
clusterrolebinding.rbac.authorization.k8s.io/prometheus-role-binding created
packageinstall.packaging.carvel.dev/prometheus created
```

10 Überprüfen Sie die Installation des Prometheus-Pakets.

```
kubectl get pkgi -A
```

11 Überprüfen Sie die Prometheus-Objekte.

```
kubectl get all -n  tanzu-system-monitoring
```

```
NAME                                                READY   STATUS    RESTARTS   AGE
pod/alertmanager-757ffd8c6c-97kqd                   1/1     Running   0          87s
pod/prometheus-kube-state-metrics-67b965c5d8-8mf4k   1/1     Running   0          87s
pod/prometheus-node-exporter-4spk9                  1/1     Running   0          87s
pod/prometheus-node-exporter-6k2rh                  1/1     Running   0          87s
pod/prometheus-node-exporter-7z9s8                  1/1     Running   0          87s
pod/prometheus-node-exporter-9d6ss                  1/1     Running   0          87s
pod/prometheus-node-exporter-csbwc                  1/1     Running   0          87s
pod/prometheus-node-exporter-qdb72                  1/1     Running   0          87s
pod/prometheus-pushgateway-dff459565-wfrz5          1/1     Running   0          86s
pod/prometheus-server-56c68567f-bjcn5               2/2     Running   0          87s

NAME                                    TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)         AGE
service/alertmanager                    ClusterIP   10.109.54.17     <none>
80/TCP          88s
service/prometheus-kube-state-metrics   ClusterIP   None             <none>        80/
TCP,81/TCP   88s
service/prometheus-node-exporter        ClusterIP   10.104.132.133   <none>
9100/TCP        88s
service/prometheus-pushgateway          ClusterIP   10.109.80.171    <none>
9091/TCP        88s
service/prometheus-server               ClusterIP   10.103.252.220   <none>
80/TCP          87s

NAME                                      DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/prometheus-node-exporter   6         6         6       6
6           <none>          88s

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/alertmanager                  1/1     1            1           88s
```

```
deployment.apps/prometheus-kube-state-metrics   1/1     1            1           88s
deployment.apps/prometheus-pushgateway          1/1     1            1           87s
deployment.apps/prometheus-server               1/1     1            1           88s

NAME                                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/alertmanager-757ffd8c6c                    1         1         1       88s
replicaset.apps/prometheus-kube-state-metrics-67b965c5d8   1         1         1       88s
replicaset.apps/prometheus-pushgateway-dff459565           1         1         1       87s
replicaset.apps/prometheus-server-56c68567f                1         1         1       88s
```

12 Überprüfen Sie die Prometheus-PVC.

```
kubectl get pvc -n tanzu-system-monitoring
```

```
NAME                STATUS   VOLUME                                     CAPACITY   ACCESS
MODES   STORAGECLASS             AGE
alertmanager        Bound    pvc-5781956b-abc4-4646-b54c-a3eda1bf140c   2Gi
RWO            vsphere-default-policy   53m
prometheus-server   Bound    pvc-9d45d7cb-6754-40a6-a4b6-f47cf6c949a9   20Gi
RWO            vsphere-default-policy   53m
```

Zugreifen auf das Prometheus-Dashboard

Führen Sie nach der Installation von Prometheus die folgenden Schritte aus, um auf das Prometheus-Dashboard zuzugreifen.

1 Stellen Sie sicher, dass der `ingress`-Abschnitt der `prometheus-data-values.yaml`-Datei mit allen erforderlichen Feldern gefüllt ist.

```
ingress:
  enabled: true
  virtual_host_fqdn: "prometheus.system.tanzu"
  prometheus_prefix: "/"
  alertmanager_prefix: "/alertmanager/"
  prometheusServicePort: 80
  alertmanagerServicePort: 80
  #! [Optional] The certificate for the ingress if you want to use your own TLS
certificate.
  #! We will issue the certificate by cert-manager when it's empty.
  tlsCertificate:
    #! [Required] the certificate
    tls.crt:
    #! [Required] the private key
    tls.key:
    #! [Optional] the CA certificate
    ca.crt:
```

2 Rufen Sie die öffentliche (externe) IP-Adresse des Contour mit Envoy-Lastausgleichsdiensts ab.

```
kubectl -n tanzu-system-ingress get all
```

3 Starten Sie die Prometheus-Webschnittstelle.

```
kubectl get httpproxy -n tanzu-system-monitoring
```

Der FQDN sollte unter der öffentlichen IP-Adresse für den Envoy-Dienst verfügbar sein.

```
NAME                   FQDN                       TLS SECRET       STATUS   STATUS
DESCRIPTION
prometheus-httpproxy   prometheus.system.tanzu   prometheus-tls   valid    Valid  HTTPProxy
```

4 Erstellen Sie einen DNS-Datensatz, der den Prometheus-FQDN zur externen IP-Adresse des Envoy-Lastausgleichsdiensts zuordnet.

5 Greifen Sie auf das Prometheus-Dashboard zu, indem Sie mit einem Browser zum Prometheus-FQDN navigieren.

prometheus-data-values.yaml

```
alertmanager:
  config:
    alertmanager_yml: "global: {}\nreceivers:\n- name: default-receiver\ntemplates:\n\
      - '/etc/alertmanager/templates/*.tmpl'\nroute:\n  group_interval: 5m\n  group_wait:\
      \ 10s\n  receiver: default-receiver\n  repeat_interval: 3h\n"
  deployment:
    containers:
      resources: {}
    podAnnotations: {}
    podLabels: {}
    replicas: 1
    rollingUpdate:
      maxSurge: null
      maxUnavailable: null
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    annotations: {}
    storage: 2Gi
    storageClassName: wcpglobalstorageprofile
  service:
    annotations: {}
    labels: {}
    port: 80
    targetPort: 9093
    type: ClusterIP
ingress:
  alertmanagerServicePort: 80
  alertmanager_prefix: /alertmanager/
  enabled: false
  prometheusServicePort: 80
  prometheus_prefix: /
  tlsCertificate:
    ca.crt: null
    tls.crt: null
    tls.key: null
```

```
  virtual_host_fqdn: prometheus.system.tanzu
kube_state_metrics:
  deployment:
    containers:
      resources: {}
    podAnnotations: {}
    podLabels: {}
    replicas: 1
  service:
    annotations: {}
    labels: {}
    port: 80
    targetPort: 8080
    telemetryPort: 81
    telemetryTargetPort: 8081
    type: ClusterIP
namespace: tanzu-system-monitoring
node_exporter:
  daemonset:
    containers:
      resources: {}
    hostNetwork: false
    podAnnotations: {}
    podLabels: {}
    updatestrategy: RollingUpdate
  service:
    annotations: {}
    labels: {}
    port: 9100
    targetPort: 9100
    type: ClusterIP
prometheus:
  config:
    alerting_rules_yml: '{}

      '
    alerts_yml: '{}

      '
    prometheus_yml: "global:\n  evaluation_interval: 1m\n  scrape_interval: 1m\n \
      \ scrape_timeout: 10s\nrule_files:\n- /etc/config/alerting_rules.yml\n- /etc/config/
recording_rules.yml\n\
      - /etc/config/alerts\n- /etc/config/rules\nscrape_configs:\n- job_name: 'prometheus'\n\
      \  scrape_interval: 5s\n  static_configs:\n  - targets: ['localhost:9090']\n\
      - job_name: 'kube-state-metrics'\n  static_configs:\n  - targets: ['prometheus-kube-
state-metrics.tanzu-system-monitoring.svc.cluster.local:8080']\n\
      \n- job_name: 'node-exporter'\n  static_configs:\n  - targets: ['prometheus-node-
exporter.tanzu-system-monitoring.svc.cluster.local:9100']\n\
      \n- job_name: 'kubernetes-pods'\n  kubernetes_sd_configs:\n  - role: pod\n \
      \ relabel_configs:\n  - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_scrape]\n\
      \    action: keep\n    regex: true\n  - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_path]\n\
      \    action: replace\n    target_label: __metrics_path__\n    regex: (.+)\n\
      \  - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
```

```
\n\
      \    action: replace\n    regex: ([^:]+)(?::\\d+)?;(\\d+)\n    replacement:\
      \ $1:$2\n    target_label: __address__\n  - action: labelmap\n    regex:
__meta_kubernetes_pod_label_(.+)\n\
      \  - source_labels: [__meta_kubernetes_namespace]\n    action: replace\n   \
      \ target_label: kubernetes_namespace\n  - source_labels: [__meta_kubernetes_pod_name]\n\
      \    action: replace\n    target_label: kubernetes_pod_name\n- job_name: kubernetes-
nodes-cadvisor\n\
      \  kubernetes_sd_configs:\n  - role: node\n  relabel_configs:\n  - action: labelmap\n\
      \    regex: __meta_kubernetes_node_label_(.+)\n  - replacement:
kubernetes.default.svc:443\n\
      \    target_label: __address__\n  - regex: (.+)\n    replacement: /api/v1/nodes/$1/
proxy/metrics/cadvisor\n\
      \    source_labels:\n    - __meta_kubernetes_node_name\n    target_label:
__metrics_path__\n\
      \  scheme: https\n  tls_config:\n    ca_file: /var/run/secrets/kubernetes.io/
serviceaccount/ca.crt\n\
      \    insecure_skip_verify: true\n  bearer_token_file: /var/run/secrets/kubernetes.io/
serviceaccount/token\n\n\
      - job_name: kubernetes-apiservers\n  kubernetes_sd_configs:\n  - role: endpoints\n\
      \  relabel_configs:\n  - action: keep\n    regex: default;kubernetes;https\n\
      \    source_labels:\n    - __meta_kubernetes_namespace\n    -
__meta_kubernetes_service_name\n\
      \    - __meta_kubernetes_endpoint_port_name\n  scheme: https\n  tls_config:\n\
      \    ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt\n
insecure_skip_verify:\
      \ true\n  bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token\n\
      alerting:\n  alertmanagers:\n  - scheme: http\n    static_configs:\n    - targets:\n\
      \      - alertmanager.tanzu-system-monitoring.svc:80\n  - kubernetes_sd_configs:\n\
      \      - role: pod\n    relabel_configs:\n    - source_labels:
[__meta_kubernetes_namespace]\n\
      \      regex: default\n      action: keep\n    - source_labels:
[__meta_kubernetes_pod_label_app]\n\
      \      regex: prometheus\n      action: keep\n    - source_labels:
[__meta_kubernetes_pod_label_component]\n\
      \      regex: alertmanager\n      action: keep\n    - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_probe]\n\
      \      regex: .*\n      action: keep\n    - source_labels:
[__meta_kubernetes_pod_container_port_number]\n\
      \      regex:\n      action: drop\n"
    recording_rules_yml: "groups:\n  - name: kube-apiserver.rules\n    interval: 3m\n\
      \    rules:\n    - expr: |2\n        (\n          (\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[1d]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[1d]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[1d]))\n\
      \            +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[1d]))\n\
      \          )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
```

```
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[1d]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[1d]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate1d\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[1h]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[1h]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[1h]))\n\
      \              +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[1h]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[1h]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[1h]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate1h\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[2h]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[2h]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[2h]))\n\
      \              +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[2h]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[2h]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[2h]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate2h\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[30m]))\n            -\n        \
      \    (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[30m]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[30m]))\n\
      \              +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[30m]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[30m]))\n        )\n\
```

```
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[30m]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate30m\n    - expr: |2\n        (\n          (\n  \
      \          # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[3d]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[3d]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[3d]))\n\
      \              +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[3d]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[3d]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[3d]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate3d\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[5m]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[5m]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[5m]))\n\
      \              +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[5m]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[5m]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=~\"LIST|GET\"}[5m]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate5m\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[6h]))\n            -\n         \
      \   (\n              (\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[6h]))\n\
      \                or\n                vector(0)\n              )\n          \
      \  +\n              sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[6h]))\n\
      \              +\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[6h]))\n\
      \            )\n          )\n          +\n          # errors\n
sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",code=~\"5..\"}[6h]))\n        )\n\
      \        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\"\
```

```
      ,verb=~\"LIST|GET\"}[6h]))\n      labels:\n        verb: read\n      record:\
      \ apiserver_request:burnrate6h\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\"}[1d]))\n           \
      \ -\n            sum(rate(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[1d]))\n          )\n          +\n\
      \          sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
      POST|PUT|PATCH|DELETE\",code=~\"5..\"}[1d]))\n        )\n        /\n       \
      \ \ sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|
DELETE\"\
      }[1d]))\n      labels:\n        verb: write\n      record:
apiserver_request:burnrate1d\n\
      \    - expr: |2\n        (\n          (\n            # too slow\n          \
      \ \
      \  sum(rate(apiserver_request_duration_seconds_count{job=\"kubernetes-apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\"}[1h]))\n            -\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[1h]))\n  \
      \ \
      \        )\n          +\n          sum(rate(apiserver_request_total{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",code=~\"5..\"}[1h]))\n        )\n        /\n\
      \        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
      POST|PUT|PATCH|DELETE\"}[1h]))\n      labels:\n        verb: write\n      record:\
      \ apiserver_request:burnrate1h\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\"}[2h]))\n           \
      \ -\n            sum(rate(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[2h]))\n          )\n          +\n\
      \          sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
      POST|PUT|PATCH|DELETE\",code=~\"5..\"}[2h]))\n        )\n        /\n       \
      \ \ sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|
DELETE\"\
      }[2h]))\n      labels:\n        verb: write\n      record:
apiserver_request:burnrate2h\n\
      \    - expr: |2\n        (\n          (\n            # too slow\n          \
      \ \
      \  sum(rate(apiserver_request_duration_seconds_count{job=\"kubernetes-apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\"}[30m]))\n            -\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[30m]))\n \
      \ \
      \         )\n          +\n          sum(rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\",code=~\"5..\"}[30m]))\n\
      \        )\n        /\n        sum(rate(apiserver_request_total{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\"}[30m]))\n      labels:\n        verb: write\n\
      \      record: apiserver_request:burnrate30m\n    - expr: |2\n        (\n  \
      \      (\n            # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\"}[3d]))\n           \
      \ -\n            sum(rate(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[3d]))\n          )\n          +\n\
      \          sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
```

```
      POST|PUT|PATCH|DELETE\",code=~\"5..\"}[3d]))\n        )\n        /\n       \
      \ sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|
DELETE\"\
      }[3d]))\n      labels:\n        verb: write\n      record:
apiserver_request:burnrate3d\n\
      \    - expr: |2\n        (\n          (\n            # too slow\n          \
      \  sum(rate(apiserver_request_duration_seconds_count{job=\"kubernetes-apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\"}[5m]))\n            -\n
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[5m]))\n  \
      \        )\n          +\n          sum(rate(apiserver_request_total{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",code=~\"5..\"}[5m]))\n        )\n        /\n\
      \        sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
      POST|PUT|PATCH|DELETE\"}[5m]))\n      labels:\n        verb: write\n      record:\
      \ apiserver_request:burnrate5m\n    - expr: |2\n        (\n          (\n   \
      \         # too slow\n
sum(rate(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\"}[6h]))\n           \
      \ -\n            sum(rate(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\",le=\"1\"}[6h]))\n          )\n          +\n\
      \          sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"\
      POST|PUT|PATCH|DELETE\",code=~\"5..\"}[6h]))\n        )\n        /\n       \
      \ sum(rate(apiserver_request_total{job=\"kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|
DELETE\"\
      }[6h]))\n      labels:\n        verb: write\n      record:
apiserver_request:burnrate6h\n\
      \    - expr: |\n        sum by (code,resource) (rate(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[5m]))\n      labels:\n        verb:\
      \ read\n      record: code_resource:apiserver_request_total:rate5m\n    - expr:\
      \ |\n        sum by (code,resource) (rate(apiserver_request_total{job=\"kubernetes-
apiservers\"\
      ,verb=~\"POST|PUT|PATCH|DELETE\"}[5m]))\n      labels:\n        verb: write\n\
      \      record: code_resource:apiserver_request_total:rate5m\n    - expr: |\n\
      \        histogram_quantile(0.99, sum by (le, resource)
(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[5m]))) > 0\n      labels:\n    \
      \    quantile: \"0.99\"\n        verb: read\n      record:
cluster_quantile:apiserver_request_duration_seconds:histogram_quantile\n\
      \    - expr: |\n        histogram_quantile(0.99, sum by (le, resource)
(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"POST|PUT|PATCH|DELETE\"}[5m]))) > 0\n      labels:\n\
      \        quantile: \"0.99\"\n        verb: write\n      record:
cluster_quantile:apiserver_request_duration_seconds:histogram_quantile\n\
      \    - expr: |2\n        sum(rate(apiserver_request_duration_seconds_sum{subresource!
=\"\
      log\",verb!~\"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT\"}[5m]))
without(instance,\
      \ pod)\n        /\n
sum(rate(apiserver_request_duration_seconds_count{subresource!=\"\
      log\",verb!~\"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT\"}[5m]))
without(instance,\
      \ pod)\n      record: cluster:apiserver_request_duration_seconds:mean5m\n  \
      \  - expr: |\n        histogram_quantile(0.99,
```

```
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",subresource!=\"log\",verb!~\"LIST|WATCH|WATCHLIST|
DELETECOLLECTION|PROXY|CONNECT\"\
      }[5m])) without(instance, pod))\n      labels:\n        quantile: \"0.99\"\n\
      \      record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile\n\
      \    - expr: |\n        histogram_quantile(0.9,
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",subresource!=\"log\",verb!~\"LIST|WATCH|WATCHLIST|
DELETECOLLECTION|PROXY|CONNECT\"\
      }[5m])) without(instance, pod))\n      labels:\n        quantile: \"0.9\"\n\
      \      record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile\n\
      \    - expr: |\n        histogram_quantile(0.5,
sum(rate(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",subresource!=\"log\",verb!~\"LIST|WATCH|WATCHLIST|
DELETECOLLECTION|PROXY|CONNECT\"\
      }[5m])) without(instance, pod))\n      labels:\n        quantile: \"0.5\"\n\
      \      record: cluster_quantile:apiserver_request_duration_seconds:histogram_quantile\n\
      \  - interval: 3m\n    name: kube-apiserver-availability.rules\n    rules:\n\
      \    - expr: |2\n        1 - (\n          (\n            # write too slow\n\
      \            sum(increase(apiserver_request_duration_seconds_count{verb=~\"\
      POST|PUT|PATCH|DELETE\"}[30d]))\n            -\n
sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"\
      POST|PUT|PATCH|DELETE\",le=\"1\"}[30d]))\n          ) +\n          (\n     \
      \       # read too slow\n
sum(increase(apiserver_request_duration_seconds_count{verb=~\"\
      LIST|GET\"}[30d]))\n            -\n            (\n              (\n        \
      \        sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"LIST|GET\"\
      ,scope=~\"resource|\",le=\"0.1\"}[30d]))\n               or\n             \
      \   \n      \   vector(0)\n             )\n             +\n
sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"\
      LIST|GET\",scope=\"namespace\",le=\"0.5\"}[30d]))\n             +\n       \
      \   \n      \       sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"LIST|GET\"\
      ,scope=\"cluster\",le=\"5\"}[30d]))\n            )\n          ) +\n        \
      \  \n      \  # errors\n          sum(code:apiserver_request_total:increase30d{code=~\"\
      5..\"} or vector(0))\n        )\n        /\n
sum(code:apiserver_request_total:increase30d)\n\
      \      labels:\n        verb: all\n      record: apiserver_request:availability30d\n\
      \    - expr: |2\n        1 - (\n
sum(increase(apiserver_request_duration_seconds_count{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\"}[30d]))\n          -\n          (\n\
      \            # too slow\n            (\n
sum(increase(apiserver_request_duration_seconds_bucket{job=\"\
      kubernetes-apiservers\",verb=~\"LIST|GET\",scope=~\"resource|\",le=\"0.1\"}[30d]))\n\
      \              or\n              vector(0)\n            )\n            +\n \
      \           sum(increase(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"LIST|GET\",scope=\"namespace\",le=\"0.5\"}[30d]))\n            +\n\
      \            sum(increase(apiserver_request_duration_seconds_bucket{job=\"kubernetes-
apiservers\"\
      ,verb=~\"LIST|GET\",scope=\"cluster\",le=\"5\"}[30d]))\n          )\n      \
      \  \n      \    +\n          # errors\n
sum(code:apiserver_request_total:increase30d{verb=\"\
      read\",code=~\"5..\"} or vector(0))\n        )\n        /\n
sum(code:apiserver_request_total:increase30d{verb=\"\
      read\"})\n      labels:\n        verb: read\n      record:
```

```
apiserver_request:availability30d\n\
      \    - expr: |2\n        1 - (\n          (\n            # too slow\n      \
      \      sum(increase(apiserver_request_duration_seconds_count{verb=~\"POST|PUT|PATCH|
DELETE\"\
      }[30d]))\n            -\n
sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"\
      POST|PUT|PATCH|DELETE\",le=\"1\"}[30d]))\n          )\n          +\n       \
      \   # errors\n          sum(code:apiserver_request_total:increase30d{verb=\"\
      write\",code=~\"5..\"} or vector(0))\n        )\n        /\n
sum(code:apiserver_request_total:increase30d{verb=\"\
      write\"})\n      labels:\n        verb: write\n      record:
apiserver_request:availability30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"LIST\",code=~\"2..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"GET\",code=~\"2..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"POST\",code=~\"2..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PUT\",code=~\"2..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PATCH\",code=~\"2..\"}[30d]))\n      record:\
      \ code_verb:apiserver_request_total:increase30d\n    - expr: |\n        sum\
      \ by (code, verb) (increase(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=\"DELETE\",code=~\"2..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"LIST\",code=~\"3..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"GET\",code=~\"3..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"POST\",code=~\"3..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PUT\",code=~\"3..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PATCH\",code=~\"3..\"}[30d]))\n      record:\
      \ code_verb:apiserver_request_total:increase30d\n    - expr: |\n        sum\
      \ by (code, verb) (increase(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=\"DELETE\",code=~\"3..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"LIST\",code=~\"4..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"GET\",code=~\"4..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
```

```
      kubernetes-apiservers\",verb=\"POST\",code=~\"4..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PUT\",code=~\"4..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PATCH\",code=~\"4..\"}[30d]))\n      record:\
      \ code_verb:apiserver_request_total:increase30d\n    - expr: |\n        sum\
      \ by (code, verb) (increase(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=\"DELETE\",code=~\"4..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"LIST\",code=~\"5..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"GET\",code=~\"5..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"POST\",code=~\"5..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PUT\",code=~\"5..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code, verb) (increase(apiserver_request_total{job=\"\
      kubernetes-apiservers\",verb=\"PATCH\",code=~\"5..\"}[30d]))\n      record:\
      \ code_verb:apiserver_request_total:increase30d\n    - expr: |\n        sum\
      \ by (code, verb) (increase(apiserver_request_total{job=\"kubernetes-apiservers\"\
      ,verb=\"DELETE\",code=~\"5..\"}[30d]))\n      record:
code_verb:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code)
(code_verb:apiserver_request_total:increase30d{verb=~\"\
      LIST|GET\"})\n      labels:\n        verb: read\n      record:
code:apiserver_request_total:increase30d\n\
      \    - expr: |\n        sum by (code)
(code_verb:apiserver_request_total:increase30d{verb=~\"\
      POST|PUT|PATCH|DELETE\"})\n      labels:\n        verb: write\n      record:\
      \ code:apiserver_request_total:increase30d\n"
    rules_yml: '{}

      '
  deployment:
    configmapReload:
      containers:
        args:
        - --volume-dir=/etc/config
        - --webhook-url=http://127.0.0.1:9090/-/reload
        resources: {}
    containers:
      args:
      - --storage.tsdb.retention.time=42d
      - --config.file=/etc/config/prometheus.yml
      - --storage.tsdb.path=/data
      - --web.console.libraries=/etc/prometheus/console_libraries2
      - --web.console.templates=/etc/prometheus/consoles
      - --web.enable-lifecycle
```

```
      resources: {}
    podAnnotations: {}
    podLabels: {}
    replicas: 1
    rollingUpdate:
      maxSurge: null
      maxUnavailable: null
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    annotations: {}
    storage: 150Gi
    storageClassName: wcpglobalstorageprofile
  service:
    annotations: {}
    labels: {}
    port: 80
    targetPort: 9090
    type: ClusterIP
pushgateway:
  deployment:
    containers:
      resources: {}
    podAnnotations: {}
    podLabels: {}
    replicas: 1
  service:
    annotations: {}
    labels: {}
    port: 9091
    targetPort: 9091
    type: ClusterIP
```

prometheus.yaml

Die Spezifikation `prometheus.yaml` verweist auf den geheimen `prometheus-data-values`-Schlüssel.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus-sa
  namespace: tkg-system

---
# temp
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
```

```
subjects:
  - kind: ServiceAccount
    name: prometheus-sa
    namespace: tkg-system

---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: prometheus
  namespace: tkg-system
spec:
  serviceAccountName: prometheus-sa
  packageRef:
    refName: prometheus.tanzu.vmware.com
    versionSelection:
      constraints: 2.45.0+vmware.1-tkg.2
  values:
  - secretRef:
      name: prometheus-data-values
```

Installieren von Grafana auf TKr für vSphere 7.x

Lesen Sie diese Anweisungen zum Installieren von Grafana auf einem TKG-Cluster, der mit TKr für vSphere 7.x bereitgestellt wird.

Voraussetzungen

Weitere Informationen hierzu finden Sie unter Workflow zum Installieren von Standardpaketen auf TKr für vSphere 7.x.

Installieren von Grafana

Installieren Sie Grafana.

1 Listet die verfügbaren Grafana-Versionen im Repository auf.

```
kubectl get packages -n tkg-system | grep grafana
```

2 Erstellen Sie den Grafana-Namespace.

```
kubectl create ns tanzu-system-dashboards
```

3 Erstellen Sie eine PSA-Bezeichnung für den Namespace.

```
kubectl label namespace tanzu-system-dashboards pod-security.kubernetes.io/
enforce=privileged
```

4 Alternativ können Sie den Grafana-Namespace und die Bezeichnung deklarativ mithilfe der Datei `ns-grafana-dashboard.yaml` erstellen.

```
apiVersion:
v1kind: Namespace
metadata:
  name: grafana-dashboard
---
apiVersion: v1
kind: Namespace
metadata:
  name: tanzu-system-dashboards
  labels:
    pod-security.kubernetes.io/enforce: privileged
```

5 Erstellen Sie `grafana-data-values.yaml`.

Weitere Informationen hierzu finden Sie unter Referenz zum Grafana-Paket.

6 Erstellen Sie einen geheimen Schlüssel mithilfe der als Eingabe dienenden Datei `grafana-data-values.yaml`.

```
kubectl create secret generic grafana-data-values --from-file=values.yaml=grafana-data-values.yaml -n tkg-system
```

```
secret/grafana-data-values created
```

7 Überprüfen Sie den geheimen Schlüssel.

```
kubectl get secrets -A
```

```
kubectl describe secret grafana-data-values -n tkg-system
```

8 Passen Sie bei Bedarf `grafana-data-values` für Ihre Umgebung an.

Informationen finden Sie unter Referenz zum Grafana-Paket.

Wenn Sie die Datenwerte aktualisieren, aktualisieren Sie den geheimen Schlüssel mit dem folgenden Befehl.

```
kubectl create secret generic grafana-data-values --from-file=values.yaml=grafana-data-values.yaml -n tkg-system -o yaml --dry-run=client | kubectl replace -f-
```

```
secret/grafana-data-values replaced
```

9 Erstellen Sie eine `grafana.yaml`-Spezifikation.

Weitere Informationen hierzu finden Sie unter Installieren von Grafana auf TKr für vSphere 7.x.

10 Installieren Sie Grafana.

```
kubectl apply -f grafana.yaml
```

```
serviceaccount/grafana-sa created
clusterrolebinding.rbac.authorization.k8s.io/grafana-role-binding created
packageinstall.packaging.carvel.dev/grafana created
```

11 Überprüfen Sie die Installation des Grafana-Pakets.

```
kubectl get pkgi -A | grep grafana
```

12 Überprüfen Sie die Grafana-Objekte.

```
kubectl get all -n tanzu-system-dashboards
```

Zugreifen auf das Grafana-Dashboard mithilfe von Envoy LoadBalancer

Wenn der erforderliche Contour Envoy-Dienst vom Typ „LoadBalancer“ bereitgestellt wird und Sie dies in der Grafana-Konfigurationsdatei angegeben haben, rufen Sie die externe IP-Adresse des Lastausgleichsdiensts ab und erstellen Sie DNS-Datensätze für den Grafana-FQDN.

1 Rufen Sie die `External-IP`-Adresse für den Envoy-Dienst vom Typ LoadBalancer ab.

```
kubectl get service envoy -n tanzu-system-ingress
```

Die `External-IP`-Adresse sollte angezeigt werden, z. B.:

```
NAME    TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
envoy   LoadBalancer   10.99.25.220   10.195.141.17   80:30437/TCP,443:30589/TCP   3h27m
```

Alternativ können Sie die `External-IP`-Adresse mit dem folgenden Befehl abrufen.

```
kubectl get svc envoy -n tanzu-system-ingress -o
jsonpath='{.status.loadBalancer.ingress[0]}'
```

2 Zur Überprüfung der Installation der Grafana-Erweiterung aktualisieren Sie Ihre lokale `/etc/hosts`-Datei mit dem Grafana-FQDN, der der `External-IP`-Adresse des Lastausgleichsdiensts zugeordnet ist. Beispiel:

```
127.0.0.1 localhost
127.0.1.1 ubuntu
#TKG Grafana Extension with Envoy Load Balancer
10.195.141.17 grafana.system.tanzu
```

3 Greifen Sie auf das Grafana-Dashboard zu und navigieren Sie zu `https://grafana.system.tanzu`.

Da auf der Site selbstsignierte Zertifikate verwendet werden, müssen Sie möglicherweise durch eine browserspezifische Sicherheitswarnung navigieren, bevor Sie auf das Dashboard zugreifen können.

4 Erstellen Sie für den Produktionszugriff zwei CNAME-Datensätze auf einem DNS-Server, der dem Grafana-Dashboard die Lastausgleichsdienstadresse `External-IP` des Envoy-Diensts zuordnet.

Zugreifen auf das Grafana-Dashboard mithilfe von Envoy NodePort

Wenn der erforderliche Contour Envoy-Dienst vom Typ „NodePort“ bereitgestellt wird und Sie dies in der Grafana-Konfigurationsdatei angegeben haben, rufen Sie die IP-Adresse der VM eines Worker-Knotens ab und erstellen Sie DNS-Datensätze für den Grafana-FQDN.

1 Wechseln Sie den Kontext zum vSphere-Namespace, in dem der Cluster bereitgestellt wird.

```
kubectl config use-context VSPHERE-NAMESPACE
```

2 Listet die Knoten im Cluster auf.

```
kubectl get virtualmachines
```

3 Wählen Sie einen der Worker-Knoten aus und beschreiben Sie ihn mit dem folgenden Befehl.

```
kubectl describe virtualmachines tkgs-cluster-X-workers-9twdr-59bc54dc97-kt4cm
```

4 Suchen Sie die IP-Adresse der virtuellen Maschine, z. B. `Vm Ip: 10.115.22.43`.

5 Zur Überprüfung der Installation der Grafana-Erweiterung aktualisieren Sie Ihre lokale `/etc/hosts`-Datei mit dem Grafana-FQDN, der der IP-Adresse des Worker-Knotens zugeordnet ist. Beispiel:

```
127.0.0.1 localhost
127.0.1.1 ubuntu
# TKG Grafana with Envoy NodePort
10.115.22.43 grafana.system.tanzu
```

6 Greifen Sie auf das Grafana-Dashboard zu und navigieren Sie zu `https://grafana.system.tanzu`.

Da auf der Site selbstsignierte Zertifikate verwendet werden, müssen Sie möglicherweise durch eine browserspezifische Sicherheitswarnung navigieren, bevor Sie auf das Dashboard zugreifen können.

grafana-data-values.yaml

Nachfolgend finden Sie die folgende Beispieldatei: `grafana-data-values.yaml`.

```
namespace: tanzu-system-dashboards
grafana:
  pspNames: "vmware-system-restricted"
  deployment:
```

```
    replicas: 1
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 2Gi
    storageClassName: wcpglobalstorageprofile
  secret:
    admin_user: YWRtaW4=
    admin_password: YWRtaW4=
    type: Opaque
  service:
    port: 80
    targetPort: 3000
    type: LoadBalancer
ingress:
  enabled: true
  prefix: /
  servicePort: 80
  virtual_host_fqdn: grafana.system.tanzu
```

grafana.yaml

Nachfolgend finden Sie die folgende Beispielspezifikation: `grafana.yaml`. Aktualisieren Sie die Paketversion nach Bedarf.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: grafana-sa
  namespace: tkg-system
  annotations:
    pod-security.kubernetes.io/enforce: "privileged"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: grafana-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: grafana-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: grafana
  namespace: tkg-system
spec:
  serviceAccountName: grafana-sa
  packageRef:
```

```
    refName: grafana.tanzu.vmware.com
    versionSelection:
      constraints: 10.0.1+vmware.1-tkg.2 #PKG-VERSION
  values:
  - secretRef:
      name: grafana-data-values
```

Installieren von Harbor auf TKr für vSphere 7.x

Befolgen Sie diese Anweisungen zum Installieren von Harbor auf einem TKG-Cluster, der mit einer TKr für vSphere 7.x bereitgestellt wurde.

Voraussetzungen

Weitere Informationen hierzu finden Sie unter Workflow zum Installieren von Standardpaketen auf TKr für vSphere 7.x.

Harbor erfordert eingehenden HTTP/S-Datenverkehr. Harbor-Dienste werden über einen Envoy-Dienst im Contour-Paket zur Verfügung gestellt. Stellen Sie als Voraussetzung das Contour-Paket bereit. Weitere Informationen finden Sie unter Installieren von Contour auf TKr für vSphere 7.x.

- Wenn Sie ein NSX-Netzwerk für den Supervisor verwenden, erstellen Sie einen Envoy-Dienst vom Typ „LoadBalancer“.
- Wenn Sie ein vSphere vDS-Netzwerk für den Supervisor verwenden, erstellen Sie je nach Umgebung und Anforderungen einen Envoy-Dienst vom Typ „LoadBalancer“ oder „NodePort“.

Die Harbor-Erweiterung erfordert DNS. Fügen Sie zu Test- und Prüfzwecken die Harbor- und Notar-FQDNs zu Ihrer lokalen `/etc/hosts`-Datei hinzu. Die folgenden Anweisungen beschreiben, wie Sie dies tun.

In der Produktion erfordert Harbor eine DNS-Zone entweder auf einem lokalen DNS-Server, wie z. B. BIND, oder in einer Public Cloud, wie z. B. AWS Route 53 oder Azure DNS. Nachdem Sie DNS eingerichtet haben, installieren Sie die ExternalDNS-Erweiterung, um die Harbor-FQDNs automatisch bei einem DNS-Server zu registrieren. Weitere Informationen finden Sie unter Installieren von ExternalDNS auf TKr für vSphere 7.x.

Installieren von Harbor

Führen Sie die folgenden Schritte aus, um die Harbor-Registrierung mithilfe des Standardpakets zu installieren.

1 Listen Sie die verfügbaren Harbor-Versionen im Repository auf.

```
kubectl get packages -n tkg-system | grep harbor
```

2 Erstellen Sie eine `harbor.yaml`-Spezifikation.

```
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  name: harbor-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: habor-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: harbor-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: harbor
  namespace: tkg-system
spec:
  serviceAccountName: harbor-sa
  packageRef:
    refName: harbor.tanzu.vmware.com
    versionSelection:
      constraints: 2.7.1+vmware.1-tkg.1 #PKG-VERSION
  values:
  - secretRef:
      name: harbor-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: harbor-data-values
  namespace: harbor-registry
stringData:
  values.yml: |
    namespace: tanzu-system-registry
    hostname: <ENTER-HARBOR-FQDN>
    port:
      https: 443
    logLevel: info
    tlsCertificate:
      tls.crt: ""
      tls.key: ""
      ca.crt:
    tlsCertificateSecretName:
    enableContourHttpProxy: true
    harborAdminPassword: <ENTER-STRONG-PASSWORD-HERE>
    secretKey: <ENTER-SECRET-KEY>
    database:
      password: <ENTER-STRONG-PASSWORD-HERE>
      shmSizeLimit:
      maxIdleConns:
```

```
    maxOpenConns:
  exporter:
    cacheDuration:
  core:
    replicas: 1
    secret: <ENTER-SECRET>
    xsrfKey: <ENTER-XSRF-KEY-WHICH-IS-AN-ALPHANUMERIC-STRING-WITH-32-CHARS>
  jobservice:
    replicas: 1
    secret: <ENTER-SECRET>
  registry:
    replicas: 1
    secret: <ENTER-SECRET>
  trivy:
    enabled: true
    replicas: 1
    gitHubToken: ""
    skipUpdate: false
  persistence:
    persistentVolumeClaim:
      registry:
        existingClaim: ""
        storageClass: "<ENTER-STORAGE-CLASS>"
        subPath: ""
        accessMode: ReadWriteOnce
        size: 50Gi
      jobservice:
        existingClaim: ""
        storageClass: "<ENTER-STORAGE-CLASS>"
        subPath: ""
        accessMode: ReadWriteOnce
        size: 10Gi
      database:
        existingClaim: ""
        storageClass: "<ENTER-STORAGE-CLASS>"
        subPath: ""
        accessMode: ReadWriteOnce
        size: 10Gi
      redis:
        existingClaim: ""
        storageClass: "<ENTER-STORAGE-CLASS>"
        subPath: ""
        accessMode: ReadWriteOnce
        size: 10Gi
      trivy:
        existingClaim: ""
        storageClass: "<ENTER-STORAGE-CLASS>"
        subPath: ""
        accessMode: ReadWriteOnce
        size: 10Gi
  proxy:
    httpProxy:
    httpsProxy:
```

```
      noProxy: 127.0.0.1,localhost,.local,.internal
    pspNames: vmware-system-restricted
    network:
      ipFamilies: ["IPv4", "IPv6"]
```

3 Passen Sie den geheimen `harbor-data-values`-Schlüssel in der `harbor.yaml`-Spezifikation mit entsprechenden Werten für Ihre Umgebung an, einschließlich Hostname, Kennwörter, geheimer Schlüssel und Speicherklasse.

   Weitere Informationen finden Sie unter Referenz zum Harbor-Paket.

4 Installieren Sie Harbor.

```
kubectl apply -f harbor.yaml
```

5 Überprüfen Sie die Harbor-Installation.

```
kubectl get all -n harbor-registry
```

Konfigurieren von DNS für Harbor mithilfe von Envoy LoadBalancer (NSX-Netzwerk)

Wenn der erforderliche Envoy-Dienst über einen LoadBalancer verfügbar gemacht wird, rufen Sie die externe IP-Adresse des Lastausgleichsdiensts ab und erstellen Sie DNS-Datensätze für die Harbor-FQDNs.

1 Rufen Sie die `External-IP`-Adresse für den Envoy-Dienst vom Typ LoadBalancer ab.

```
kubectl get service envoy -n tanzu-system-ingress
```

   Die `External-IP`-Adresse sollte angezeigt werden, z. B.:

```
NAME    TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
envoy   LoadBalancer   10.99.25.220   10.195.141.17   80:30437/TCP,443:30589/TCP   3h27m
```

   Alternativ können Sie die `External-IP`-Adresse mit dem folgenden Befehl abrufen.

```
kubectl get svc envoy -n tanzu-system-ingress -o
jsonpath='{.status.loadBalancer.ingress[0]}'
```

2 Um die Installation der Harbor-Erweiterung zu überprüfen, aktualisieren Sie Ihre lokale `/etc/hosts`-Datei mit den Harbor- und Notariats-FQDNs, die der `External-IP`-Adresse des Lastausgleichsdiensts zugeordnet sind. Beispiel:

```
127.0.0.1 localhost
127.0.1.1 ubuntu
#TKG Harbor with Envoy Load Balancer IP
10.195.141.17 core.harbor.domain
10.195.141.17 core.notary.harbor.domain
```

3 Um die Installation der Harbor-Erweiterung zu überprüfen, melden Sie sich bei Harbor an.

4 Erstellen Sie zwei CNAME-Datensätze auf einem DNS-Server, die die `External-IP`-Adresse des Lastausgleichsdiensts für den Envoy-Dienst dem Harbor-FQDN und dem Notariats-FQDN zuordnen.

5 Installieren Sie die externe DNS-Erweiterung.

Konfigurieren von DNS für Harbor mithilfe von Envoy NodePort (vDS-Netzwerk)

Wenn der erforderliche Envoy-Dienst über einen NodePort verfügbar gemacht wird, rufen Sie die IP-Adresse der virtuellen Maschine eines Worker-Knotens ab und erstellen Sie DNS-Datensätze für die Harbor-FQDNs.

Hinweis Um NodePort zu verwenden, müssen Sie den korrekten `port.https`-Wert in der `harbor-data-values.yaml`-Datei angegeben haben.

1 Wechseln Sie den Kontext zum vSphere-Namespace, in dem der Cluster bereitgestellt wird.

```
kubectl config use-context VSPHERE-NAMESPACE
```

2 Listet die Knoten im Cluster auf.

```
kubectl get virtualmachines
```

3 Wählen Sie einen der Worker-Knoten aus und beschreiben Sie ihn mit dem folgenden Befehl.

```
kubectl describe virtualmachines tkg2-cluster-X-workers-9twdr-59bc54dc97-kt4cm
```

4 Suchen Sie die IP-Adresse der virtuellen Maschine, z. B. `Vm Ip: 10.115.22.43`.

5 Um die Installation der Harbor-Erweiterung zu überprüfen, aktualisieren Sie Ihre lokale `/etc/hosts`-Datei mit den Harbor- und Notariats-FQDNs, die der IP-Adresse des Worker-Knotens zugeordnet sind. Beispiel:

```
127.0.0.1 localhost
127.0.1.1 ubuntu
#TKG Harbor with Envoy NodePort
10.115.22.43 core.harbor.domain
10.115.22.43 core.notary.harbor.domain
```

6 Um die Installation der Harbor-Erweiterung zu überprüfen, melden Sie sich bei Harbor an.

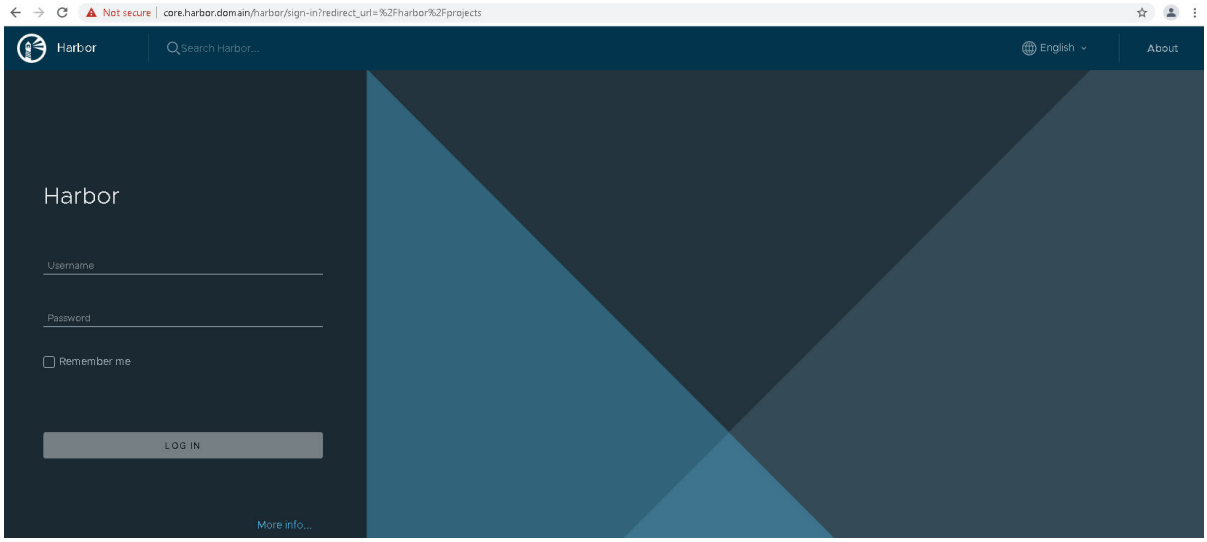
7 Erstellen Sie zwei CNAME-Datensätze auf einem DNS-Server, die die IP-Adresse des Worker-Knotens für den Envoy-Dienst dem Harbor-FQDN und dem Notariats-FQDN zuordnen.

8 Installieren Sie die externe DNS-Erweiterung.

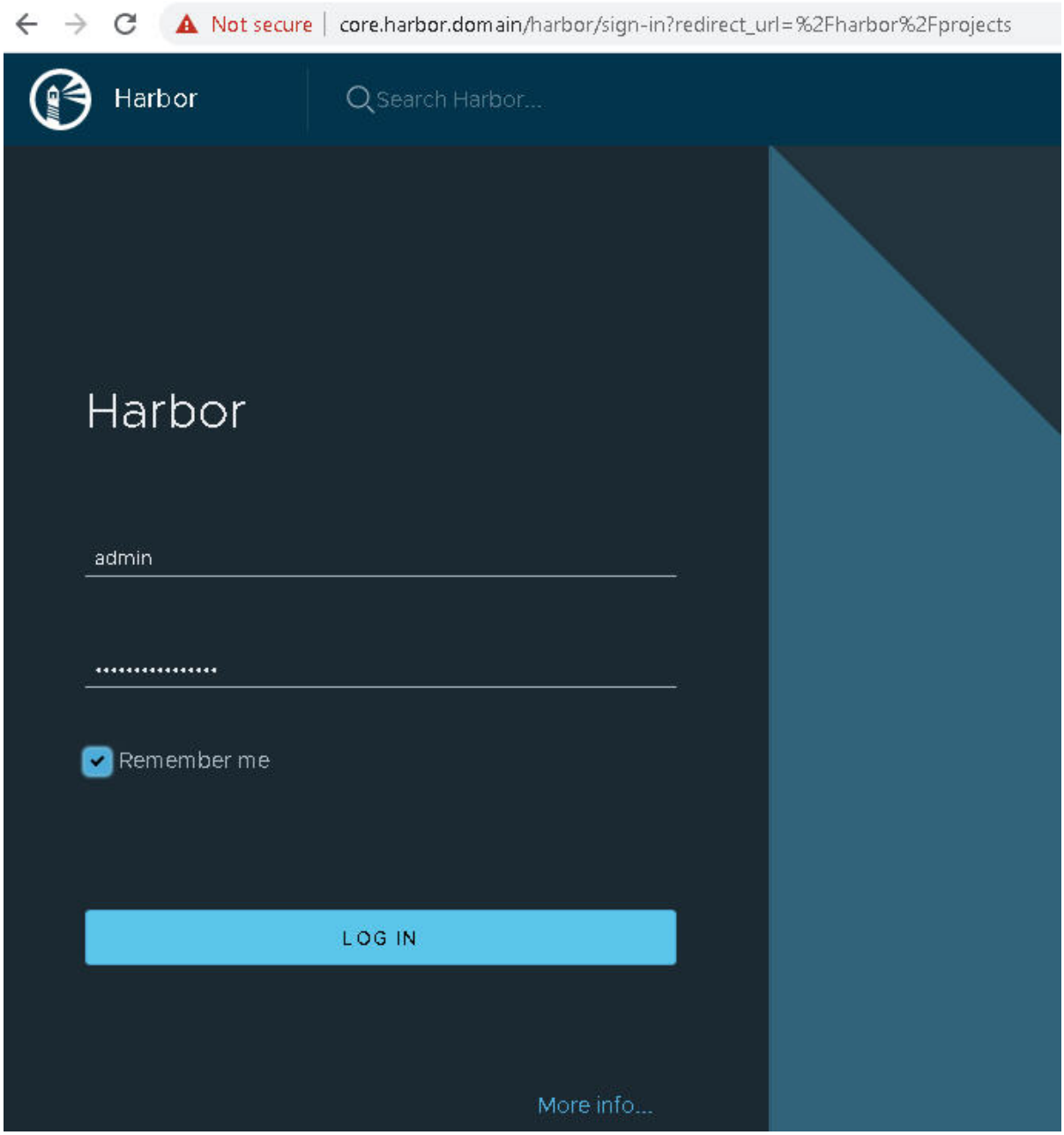
Anmelden bei der Harbor-Webschnittstelle

Sobald Harbor installiert und konfiguriert ist, melden Sie sich an und verwenden Sie es.

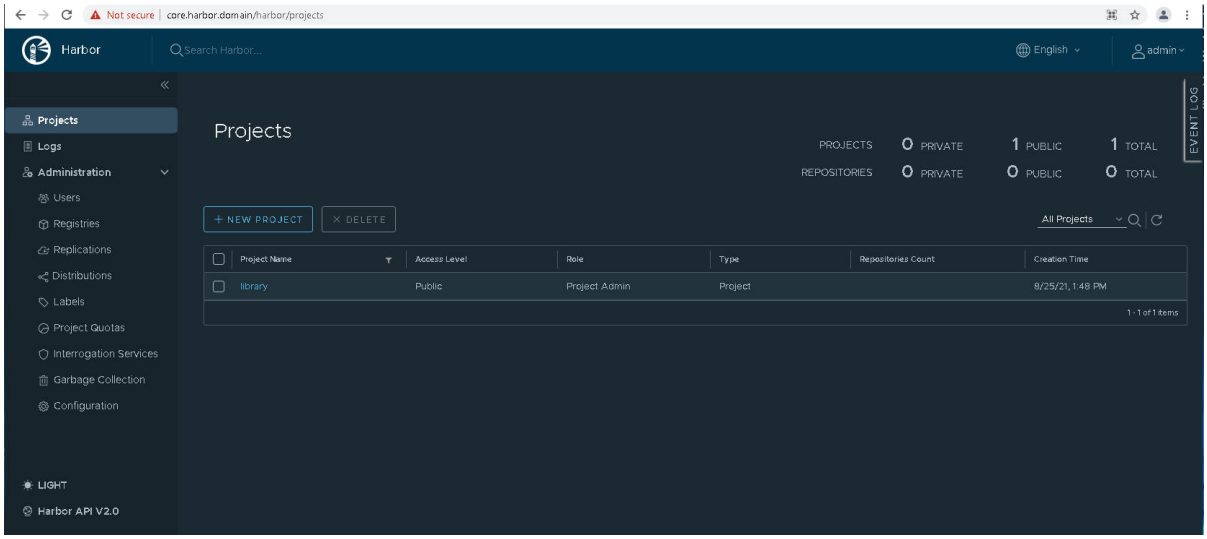
1 Greifen Sie unter `https://core.harbor.domain` oder dem von Ihnen verwendeten Hostnamen auf die Webschnittstelle der Harbor-Registrierung zu.



2 Melden Sie sich bei Harbor mit dem Benutzernamen **`admin`** und dem generierten Kennwort an, das Sie in die `harbor-data-values.yaml`-Datei eingegeben haben.



3 Überprüfen Sie, ob Sie über die Harbor-Benutzeroberfläche auf den Host zugreifen können.



4 Rufen Sie das Harbor-CA-Zertifikat ab.

   Klicken Sie in der Harbor-Benutzeroberfläche auf **Projekte > Bibliothek**, oder erstellen Sie ein **Neues Projekt**.

   Klicken Sie auf **Registrierungszertifikat** und laden Sie das Harbor-CA-Zertifikat (`ca.crt`) herunter.

5 Fügen Sie das Harbor-CA-Zertifikat zum Trust Store des Docker-Clients hinzu, damit Sie Container-Images in die Harbor-Registrierung verschieben und von dort abrufen können. Weitere Informationen finden Sie unter Kapitel 14 Verwenden privater Registrierungen mit TKG-Dienstclustern.

6 Weitere Informationen zur Verwendung von Harbor finden Sie in der Harbor-Dokumentation.

Bereitstellen von Arbeitslasten auf TKG-Dienst-Clustern

12

Sie können Anwendungsarbeitslasten für TKG-Dienst-Cluster mithilfe von Pods, Diensten, persistenten Volumes und Ressourcen auf höherer Ebene (z. B. Bereitstellungen und Replikat-Sets) implementieren.

Lesen Sie als Nächstes die folgenden Themen:

- Pod-Bereitstellung mit Lastausgleichsdienst
- Lastausgleichsdienst mit statischer IP
- Ingress unter Verwendung von Nginx
- Ingress unter Verwendung von Contour
- Verwenden von Speicherklassen für persistente Volumes
- Dynamisches Erstellen dauerhafter Speicher-Volumes
- Statisches Erstellen persistenter Speicher-Volumes
- Bereitstellen der Guestbook-Anwendung in einem TKG-Cluster
- YAML für Guestbook-Anwendung
- Bereitstellen einer StatefulSet-Anwendung über vSphere-Zonen mit einem Volume-Anhang mit später Bindung

Pod-Bereitstellung mit Lastausgleichsdienst

Um externen Datenverkehr an Pods weiterzuleiten, die in einem TKG 2.0-Cluster ausgeführt werden, erstellen Sie einen Dienst vom Typ „LoadBalancer“. Der Lastausgleichsdienst macht eine öffentliche IP-Adresse verfügbar, von der eingehender Datenverkehr an Pods weitergeleitet wird.

Sie können einen externen Load Balancer für Kubernetes-Pods bereitstellen, die als Dienste verfügbar gemacht werden. Beispielsweise können Sie einen Nginx-Container bereitstellen und als Kubernetes-Dienst vom Typ „LoadBalancer“ zur Verfügung stellen.

Voraussetzungen

- Sehen Sie sich die Details zum Diensttyp „LoadBalancer“ in der Dokumentation zu Kubernetes an.
- Stellen Sie einen TKG-Cluster bereit.

- Stellen Sie eine Verbindung mit dem TKG-Cluster her.

Verfahren

1 Erstellen Sie die folgende `nginx-lbsvc.yaml`-YAML-Datei.

Diese YAML-Datei definiert einen Kubernetes-Dienst vom Typ „LoadBalancer“ und stellt einen Nginx-Container als externen Load Balancer für den Dienst bereit.

```
kind: Service
apiVersion: v1
metadata:
  name: srvclb-ngnx
spec:
  selector:
    app: hello
    tier: frontend
  ports:
  - protocol: "TCP"
    port: 80
    targetPort: 80
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: loadbalancer
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
        tier: frontend
    spec:
      containers:
      - name: nginx
        image: "nginxdemos/hello"
```

2 Wenden Sie die YAML an.

```
kubectl apply -f nginx-lbsvc.yaml
```

3 Überprüfen Sie die Bereitstellung des Nginx-Diensts.

```
kubectl get services
```

`srvclb-ngnx` ist mit einer externen und internen IP-Adresse ausgestattet.

```
NAME          TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)        AGE
srvclb-ngnx   LoadBalancer   10.11.12.19      10.19.15.89     80:30818/TCP   18m
```

4 Geben Sie über einen Browser die externe IP-Adresse für den Nginx-LoadBalancer-Dienst ein.

Sie sehen das Banner mit der `NGINX`-Meldung und Details des Lastausgleichsdiensts.

Lastausgleichsdienst mit statischer IP

Sie können einen Kubernetes-Dienst vom Typ LoadBalancer für die Verwendung einer statischen IP-Adresse konfigurieren. Beachten Sie bei der Implementierung dieser Funktion eine wichtige Sicherheitsüberlegung und die Anleitung für die Cluster-Härtung. Diese stellen eine wichtige Mindestanforderungen an die Komponenten dar.

Verwenden einer statischen IP-Adresse für einen Dienst vom Typ LoadBalancer

Wenn Sie einen Kubernetes-Dienst von Typ LoadBalancer definieren, erhalten Sie in der Regel eine vom Lastausgleichsdienst zugewiesene flüchtige IP-Adresse.

Alternativ können Sie eine statische IP-Adresse für den Lastausgleichsdienst angeben. Beim Erstellen des Diensts wird die Instanz des Lastausgleichsdiensts mit der von Ihnen zugewiesenen statischen IP-Adresse bereitgestellt.

In folgendem Beispieldienst wird die Konfiguration eines unterstützten Lastausgleichsdiensts mit einer statischen IP-Adresse dargestellt. In die Dienstspezifikation nehmen Sie den Parameter `loadBalancerIP` und einen IP-Adresswert auf, der als `10.11.12.49` in diesem Beispiel angegeben wird.

```
kind: Service
apiVersion: v1
metadata:
  name: load-balancer-service-with-static-ip
spec:
  selector:
    app: hello-world
    tier: frontend
  ports:
  - protocol: "TCP"
    port: 80
    targetPort: 80
  type: LoadBalancer
  loadBalancerIP: 10.11.12.49
```

Für den NSX Advanced Load Balancer verwenden Sie eine IP-Adresse aus dem IPAM-Pool, der für den Lastausgleichsdienst bei dessen Installation konfiguriert wurde. Wenn der Dienst erstellt und die statische IP-Adresse zugewiesen wurde, markiert der Lastausgleichsdienst ihn als zugeteilt und verwaltet den Lebenszyklus der IP-Adresse wie eine flüchtige IP-Adresse. Wenn der Dienst folglich entfernt wird, wird die Zuweisung der IP-Adresse aufgehoben, und die IP-Adresse kann neu zugeteilt werden.

Für den NSX-T Load Balancer stehen zwei Optionen zur Verfügung. Der Standardmechanismus ist mit dem NSX Advanced Load Balancer identisch: Verwenden Sie eine IP-Adresse aus dem IP-Pool, der für den Lastausgleichsdienst bei dessen Installation konfiguriert wurde. Wenn die statische IP-Adresse zugewiesen wird, markiert der Lastausgleichsdienst die Adresse automatisch als zugewiesen und verwaltet deren Lebenszyklus.

Die zweite NSX-T-Option besteht in der manuellen Vorabzuteilung der statischen IP-Adresse. In diesem Fall verwenden Sie eine IP-Adresse, die nicht Teil des dem Lastausgleichsdienst zugewiesenen externen Lastausgleichsdienst-IP-Pools ist, sondern aus einem dynamischen IP-Pool stammt. In diesem Fall verwalten Sie die Zuteilung und den Lebenszyklus der IP-Adresse manuell mithilfe von NSX Manager.

Wichtige Sicherheitsüberlegung und Härtungsanforderung

Bei Verwendung dieser Funktion ist ein potenzielles Sicherheitsproblem zu beachten. Wenn ein Entwickler den Wert `Service.status.loadBalancerIP` patchen kann, ist er unter Umständen in der Lage, den für die gepatchte IP-Adresse bestimmten Datenverkehr im Cluster zu übernehmen. Wenn insbesondere eine Rolle oder ClusterRole mit der Berechtigung `patch` an einen Dienst oder ein Benutzerkonto auf einem Cluster gebunden ist, auf dem diese Funktion implementiert ist, kann der Besitzer dieses Kontos seine eigenen Anmeldedaten verwenden, um `kubectl`-Befehle auszugeben und die dem Lastausgleichsdienst zugewiesene statische IP-Adresse zu ändern.

Um bei Verwendung der statischen IP-Zuteilung für einen Lastausgleichsdienst die möglichen Auswirkungen auf die Sicherheit zu vermeiden, müssen Sie jeden Cluster absichern, auf dem Sie diese Funktion implementieren. Hierzu darf die Rolle oder ClusterRole, die Sie für einen beliebigen Entwickler definieren, das Verb `patch` für `apiGroups:""` und `resources: services/status` nicht zulassen. Der beispielhafte Rollenausschnitt zeigt, was Sie bei der Implementierung dieser Funktion nicht tun sollten.

PATCH NICHT ZULASSEN

```
- apiGroups:
  - ""
  resources:
  - services/status
  verbs:
  - patch
```

Um zu überprüfen, ob ein Entwickler über Patch-Berechtigungen verfügt, führen Sie den folgenden Befehl als entsprechender Benutzer aus:

```
kubectl --kubeconfig <KUBECONFIG> auth can-i patch service/status
```

Wenn durch den Befehl `yes` ausgegeben wird, verfügt der Benutzer über Patch-Berechtigungen. Weitere Informationen finden Sie unter Überprüfen des API-Zugriffs in der Kubernetes-Dokumentation.

Informationen zum Gewähren von Entwicklerzugriff auf einen Cluster finden Sie unter Gewähren von vCenter SSO-Zugriff auf TKG-Dienst-Cluster für Entwickler. Ein Beispiel einer anzupassenden Rollenvorlage finden Sie unter Anwenden der standardmäßigen Pod-Sicherheitsrichtlinie auf TKG-Dienstcluster. Ein Beispiel zur Einschränkung des Clusterzugriffs finden Sie unter https://kubernetes.io/docs/reference/access-authn-authz/rbac/#role-example.

Ingress unter Verwendung von Nginx

Eine Kubernetes-Ingress-Ressource stellt HTTP- oder HTTPS-Routing von außerhalb des Clusters zu einem oder mehreren Diensten innerhalb des Clusters bereit. TKG-Cluster unterstützen Ingress durch Drittanbietercontroller wie etwa Nginx.

Dieses Lernprogramm veranschaulicht, wie Sie einen Kubernetes-Ingress-Dienst für das Routing von externem Datenverkehr zu Diensten in einem Tanzu Kubernetes-Cluster auf der Basis von NGINX bereitstellen. Für Ingress-Dienste ist jeweils ein Ingress-Controller erforderlich. Der NGINX-Ingress-Controller wird mit Helm installiert. „Helm“ ist ein Paketmanager für Kubernetes.

Hinweis Für die Durchführung dieser Aufgabe gibt es mehrere Möglichkeiten. Die hier genannten Schritte beschreiben einen möglichen Ansatz. Andere Ansätze sind möglicherweise für Sie in ihrer jeweiligen Umgebung besser geeignet.

Voraussetzungen

- Weitere Informationen finden Sie in der Dokumentation zu Kubernetes unter der Ressource Ingress.
- Lesen Sie die Dokumentation zum Ingress-Controller Nginx.
- Stellen Sie einen TKG-Cluster bereit.
- Aktivieren Sie bei Bedarf die Pod-Sicherheitsrichtlinie.
- Stellen Sie eine Verbindung mit dem TKG-Cluster her.

Verfahren

1. Installieren Sie Helm und beachten Sie dabei die Dokumentation.
2. Installieren Sie den NGINX-Ingress-Controller mit Helm.

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm install ingress-nginx ingress-nginx/ingress-nginx
```

3 Vergewissern Sie sich, dass der Nginx-Ingress-Controller als Dienst vom Typ LoadBalancer bereitgestellt wird.

```
kubectl get services
```

```
NAME                                 TYPE           CLUSTER-IP     EXTERNAL-IP
PORT(S)                      AGE
ingress-nginx-controller             LoadBalancer   10.16.18.20    10.19.14.76   80:30635/
TCP,443:30873/TCP   59m
ingress-nginx-controller-admission   ClusterIP      10.87.41.25    <none>
443/TCP                      59m
```

4 Pingen Sie den Load Balancer mithilfe der externen IP-Adresse an.

```
ping 10.19.14.76
```

```
Pinging 10.19.14.76 with 32 bytes of data:
Reply from 10.19.14.76: bytes=32 time<1ms TTL=62
Reply from 10.19.14.76: bytes=32 time=1ms TTL=62
```

5 Vergewissern Sie sich, dass der Nginx-Ingress-Controller ausgeführt wird.

```
kubectl get pods
```

```
NAME                                        READY   STATUS    RESTARTS   AGE
ingress-nginx-controller-7c6c46898c-v6blt   1/1     Running   0          76m
```

6 Erstellen Sie eine Ingress-Ressource mit einer Ingress-Regel und einem Pfad namens `ingress-hello.yaml`.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-hello
spec:
  rules:
  - http:
      paths:
      - path: /hello
        backend:
          serviceName: hello
          servicePort: 80
```

Achtung Die Kubernetes-API `networking.k8s.io/v1beta1` ist ab Kubernetes v1.22 veraltet. Die zu verwendende neue API ist `networking.k8s.io/v1` und beinhaltet mehrere Manifeständerungen, die hier dokumentiert sind: https://kubernetes.io/docs/reference/using-api/deprecation-guide/.

7 Stellen Sie die `ingress-hello`-Ressource bereit.

```
kubectl apply -f ingress-hello.yaml
```

```
ingress.networking.k8s.io/ingress-hello created
```

8 Vergewissern Sie sich, dass die Ingress-Ressource bereitgestellt wird.

Beachten Sie, dass die IP-Adresse der externen IP des Ingress-Controllers zugeordnet ist.

```
kubectl get ingress
```

```
NAME            CLASS    HOSTS   ADDRESS         PORTS   AGE
ingress-hello   <none>   *       10.19.14.76     80      51m
```

9 Erstellen Sie eine Hello-Test-App und einen Dienst namens `ingress-hello-test.yaml`.

```
kind: Service
apiVersion: v1
metadata:
  name: hello
spec:
  selector:
    app: hello
    tier: backend
  ports:
  - protocol: TCP
    port: 80
    targetPort: http
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello
      tier: backend
      track: stable
  template:
    metadata:
      labels:
        app: hello
        tier: backend
        track: stable
    spec:
      containers:
        - name: hello
```

```
          image: "gcr.io/google-samples/hello-go-gke:1.0"
          ports:
            - name: http
              containerPort: 80
```

10 Stellen Sie die `ingress-hello-test`-Ressource bereit.

```
kubectl apply -f ingress-hello-test.yaml
```

```
service/hello created
deployment.apps/hello created
```

11 Vergewissern Sie sich, dass die `hello`-Bereitstellung verfügbar ist.

```
kubectl get deployments
```

```
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
hello                       3/3     3            3           4m59s
ingress-nginx-controller    1/1     1            1           3h39m
```

12 Rufen Sie die öffentliche IP-Adresse des Lastausgleichsdiensts ab, der vom Nginx-Ingress-Controller verwendet wird.

```
kubectl get ingress
```

```
NAME            CLASS    HOSTS   ADDRESS          PORTS   AGE
ingress-hello   <none>   *       10.19.14.76      80      13m
```

13 Navigieren Sie in einem Browser zur öffentlichen IP-Adresse und schließen Sie den Ingress-Pfad ein.

```
http://10.19.14.76/hello
```

Die Meldung „Hallo“ wird zurückgegeben.

```
{"message":"Hello"}
```

Ergebnisse

Auf die Backend-App, deren Frontend der im Cluster ausgeführte Dienst ist, wird extern durch den Browser über den Ingress-Controller zugegriffen. Dabei wird die externe IP-Adresse des Lastausgleichsdiensts verwendet.

Ingress unter Verwendung von Contour

Eine Kubernetes-Ingress-Ressource stellt HTTP- oder HTTPS-Routing von außerhalb des Clusters zu einem oder mehreren Diensten innerhalb des Clusters bereit. TKG-Cluster unterstützen Ingress durch Drittanbietercontroller wie etwa Contour.

Dieses Lernprogramm veranschaulicht, wie Sie den Contour-Ingress-Controller für das Routing von externem Datenverkehr zu Diensten in einem TKG-Cluster bereitstellen. Contour ist ein Open Source-Projekt, an dem VMware beteiligt ist.

Voraussetzungen

- Weitere Informationen finden Sie in der Dokumentation zu Kubernetes unter der Ressource Ingress.
- Überprüfen Sie den Contour-Ingress-Controller.
- Stellen Sie einen TKG-Cluster bereit.
- Stellen Sie eine Verbindung mit dem TKG-Cluster her.

Verfahren

1 Erstellen Sie ein ClusterRoleBinding, mit dem Dienstkonten alle Ressourcen des Clusters verwalten können.

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding
--clusterrole=psp:vmware-system-privileged --group=system:authenticated
```

Hinweis Wenn eine strengere Sicherheit erforderlich ist, wenden Sie ein RoleBinding auf den `projectcontour`-Namespace an. Weitere Informationen hierzu finden Sie unter Anwenden der standardmäßigen Pod-Sicherheitsrichtlinie auf TKG-Dienstcluster.

2 Erstellen Sie einen Namespace mit dem Namen `projectcontour`.

Dies ist der Standard-Namespace für die Bereitstellung des Contour-Ingress-Controllers.

```
kubectl create ns projectcontour
```

3 Laden Sie die neueste YAML-Datei des Contour-Ingress-Controllers herunter: Contour-Ingress-Bereitstellung.

4 Öffnen Sie die Datei `contour.yaml` mit einem Texteditor.

5 Kommentieren Sie die folgenden beiden Zeilen aus, indem Sie jeder Zeile das Symbol `#` voranstellen:

Zeile 1632:

```
# service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp
```

Zeile 1634:

```
# externalTrafficPolicy: Local
```

6 Stellen Sie Contour bereit, indem Sie die Datei `contour.yaml` anwenden.

```
kubectl apply -f contour.yaml
```

7 Vergewissern Sie sich, dass der Contour-Ingress-Controller und der Envoy-Lastausgleichsdienst bereitgestellt werden.

```
kubectl get services -n projectcontour
```

```
NAME      TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)                      AGE
contour   ClusterIP      198.63.146.166   <none>          8001/TCP                     120m
envoy     LoadBalancer   198.48.52.47     192.168.123.5   80:30501/TCP,443:30173/TCP   120m
```

8 Vergewissern Sie sich, dass die Contour- und Envory-Pods ausgeführt werden.

```
kubectl get pods -n projectcontour
```

```
NAME                       READY   STATUS      RESTARTS   AGE
contour-7966d6cdbf-skqfl   1/1     Running     1          21h
contour-7966d6cdbf-vc8c7   1/1     Running     1          21h
contour-certgen-77m2n      0/1     Completed   0          21h
envoy-fsltp                1/1     Running     0          20h
```

9 Pingen Sie den Load Balancer mithilfe der externen IP-Adresse an.

```
ping 192.168.123.5
```

```
PING 192.168.123.5 (192.168.123.5) 56(84) bytes of data.
64 bytes from 192.168.123.5: icmp_seq=1 ttl=62 time=3.50 ms
```

10 Erstellen Sie eine Ingress-Ressource mit dem Namen `ingress-nihao.yaml`.

Erstellen Sie die YAML.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-nihao
spec:
  rules:
  - http:
      paths:
```

```
      - path: /nihao
        backend:
          serviceName: nihao
          servicePort: 80
```

Wenden Sie die YAML an.

```
kubectl apply -f ingress-nihao.yaml
```

Vergewissern Sie sich, dass die Ingress-Ressource erstellt wurde.

```
kubectl get ingress
```

Die externe IP-Adresse für den Envoy LoadBalancer (`192.168.123.5` in diesem Beispiel) wird vom Ingress-Objekt verwendet.

```
NAME            CLASS    HOSTS   ADDRESS         PORTS   AGE
ingress-nihao   <none>   *       192.168.123.5     80      17s
```

11 Stellen Sie einen Testdienst mit einer Backend-Anwendung bereit.

Erstellen Sie die folgende YAML-Datei mit dem Namen `ingress-nihao-test.yaml`:

```
kind: Service
apiVersion: v1
metadata:
  name: nihao
spec:
  selector:
    app: nihao
    tier: backend
  ports:
  - protocol: TCP
    port: 80
    targetPort: http
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nihao
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nihao
      tier: backend
      track: stable
  template:
    metadata:
      labels:
        app: nihao
        tier: backend
```

```
        track: stable
    spec:
      containers:
        - name: nihao
          image: "gcr.io/google-samples/hello-go-gke:1.0"
          ports:
            - name: http
              containerPort: 80
```

Wenden Sie die YAML an.

```
kubectl apply -f ingress-nihao-test.yaml
```

Überprüfen Sie, ob der `nihao`-Server erstellt wird.

```
kubectl get services
```

```
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
nihao        ClusterIP   10.14.21.22     <none>        80/TCP     15s
```

Vergewissern Sie sich, dass die Backend-Bereitstellung erstellt wurde.

```
kubectl get deployments
```

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nihao   3/3     3            3           2m25s
```

Vergewissern Sie sich, dass die Backend-Pods vorhanden sind.

```
kubectl get pods
```

```
NAME                     READY   STATUS    RESTARTS   AGE
nihao-8646584495-9nm8x   1/1     Running   0          106s
nihao-8646584495-vscm5   1/1     Running   0          106s
nihao-8646584495-zcsdq   1/1     Running   0          106s
```

12 Rufen Sie die öffentliche IP-Adresse des Lastausgleichsdiensts ab, der vom Contour-Ingress-Controller verwendet wird.

```
kubectl get ingress
```

```
NAME            CLASS    HOSTS   ADDRESS         PORTS   AGE
ingress-nihao   <none>   *       10.19.14.76     80      13m
```

13 Navigieren Sie in einem Browser zur öffentlichen IP-Adresse und schließen Sie den Ingress-Pfad ein.

```
http://10.19.14.76/nihao
```

Die Meldung „Hallo“ wird zurückgegeben.

```
{"message":"Hello"}
```

Ergebnisse

Auf die Backend-App, deren Frontend der im Cluster ausgeführte Dienst ist, wird extern durch den Browser über den Ingress-Controller zugegriffen. Dabei wird die externe IP-Adresse des Lastausgleichsdiensts verwendet.

Verwenden von Speicherklassen für persistente Volumes

Eine einem vSphere-Namespace zugewiesene vSphere-Speicherrichtlinie liefert zwei Editionen einer Speicherklasse, die für die Verwendung mit dauerhaften Volumes verfügbar ist. Welche Edition Sie auswählen, richtet sich nach Ihren Anforderungen.

Zwei Editionen einer für die Verwendung verfügbaren Speicherklasse

Wenn Sie einem vSphere-Namespace eine vSphere-Speicherrichtlinie zuweisen, erstellt das System eine übereinstimmende Kubernetes-Speicherklasse in diesem vSphere-Namespace. Diese Speicherklasse wird auf jeden in diesem Namespace bereitgestellten TKG-Cluster repliziert. Die Speicherklasse ist dann verfügbar, um dauerhafte Speichervolumes für Cluster-Arbeitslasten zu erstellen.

Für jede einem vSphere-Namespace zugewiesene vSphere-Speicherrichtlinie gibt es auf Supervisor eine einzelne Speicherklasse mit dem Bindungsmodus „Sofort“.

```
kubectl describe storageclass tkg-storage-policy
Name:                  tkg-storage-policy
IsDefaultClass:        No
Annotations:           cns.vmware.com/StoragePoolTypeHint=cns.vmware.com/vsan
Provisioner:           csi.vsphere.vmware.com
Parameters:            storagePolicyID=877b0f4b-b959-492a-b265-b4d460987b23
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>
```

Für jeden in diesem vSphere-Namespace bereitgestellten TKG-Cluster gibt es zwei Speicherklassen: eine, die mit der entsprechenden Speicherklasse auf Supervisor identisch ist, und eine zweite mit `*-latebinding` als Anhang zum Namen und „WaitForFirstConsumer“ als Bindungsmodus.

```
kubectl get sc
NAME                              PROVISIONER              RECLAIMPOLICY
VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
tkg-storage-policy                csi.vsphere.vmware.com   Delete
Immediate              true                   2m43s
tkg-storage-policy-latebinding    csi.vsphere.vmware.com   Delete
WaitForFirstConsumer   true                   2m43s
```

Verwenden Sie die `latebinding`-Edition der Speicherklasse, wenn Sie ein dauerhaftes Volume bereitstellen möchten, nachdem die Berechnung vom Kubernetes-Scheduler ausgewählt wurde. Weitere Informationen finden Sie unter Bindungsmodus in der Kubernetes-Dokumentation.

```
kubectl describe sc tkg-storage-policy
Name:                  tkg-storage-policy
IsDefaultClass:        No
Annotations:           <none>
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=tkg-storage-policy
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>

kubectl describe sc tkg-storage-policy-latebinding
Name:                  tkg-storage-policy-latebinding
IsDefaultClass:        No
Annotations:           <none>
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=tkg-storage-policy
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
```

Patchen einer Speicherklasse

Für TKG auf Supervisor können Sie eine Speicherklasse nicht manuell mithilfe von kubectl und YAML erstellen. Sie können eine Speicherklasse nur mithilfe des vSphere-Speicherrichtlinien-Frameworks erstellen und dann auf einen vSphere-Namespace anwenden. Das Ergebnis sind zwei entsprechende Speicherklassen auf jedem in diesem vSphere-Namespace bereitgestellten TKG-Cluster.

Auch wenn Sie eine Speicherklasse nicht manuell mithilfe von kubectl und YAML erstellen können, haben Sie die Möglichkeit, eine vorhandene Speicherklasse mithilfe von kubectl zu ändern. Dies kann erforderlich sein, wenn Sie einen TKG-Cluster ohne Angabe einer Standardspeicherklasse bereitgestellt haben und jetzt eine Anwendung mithilfe von Helm oder eines Tanzu-Pakets bereitstellen möchten, das eine Standardspeicherklasse erfordert.

Anstatt von Grund auf einen neuen Cluster mit Standardspeicher zu erstellen, können Sie die vorhandene Speicherklasse patchen und die Anmerkung „default = true“ hinzufügen, wie in der Kubernetes-Dokumentation unter Change the default StorageClass (Ändern der Standard-StorageClass) beschrieben.

Der folgende Befehl gibt beispielsweise zwei Speicherklassen zurück, die im TKG-Cluster verfügbar sind:

```
kubectl describe sc
Name:                  gc-storage-profile
IsDefaultClass:        No
Annotations:           <none>
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=gc-storage-profile
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>


Name:                  gc-storage-profile-latebinding
IsDefaultClass:        No
Annotations:           <none>
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=gc-storage-profile
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
```

Verwenden Sie den folgenden Befehl, um eine der Speicherklassen zu patchen und die Anmerkung hinzuzufügen:

```
kubectl patch storageclass gc-storage-profile -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class":"true"}}}'
storageclass.storage.k8s.io/gc-storage-profile patched
```

Überprüfen Sie die Speicherklassen erneut. Sie sehen dann, dass eine von ihnen gepatcht ist, sodass sie jetzt die Standardklasse ist.

```
kubectl describe sc
Name:                  gc-storage-profile
IsDefaultClass:        Yes
Annotations:           storageclass.kubernetes.io/is-default-class=true
```

```
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=gc-storage-profile
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>


Name:                  gc-storage-profile-latebinding
IsDefaultClass:        No
Annotations:           <none>
Provisioner:           csi.vsphere.vmware.com
Parameters:            svStorageClass=gc-storage-profile
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
```

Dynamisches Erstellen dauerhafter Speicher-Volumes

Sie können ein dauerhaftes Speicher-Volume mithilfe einer vorhandenen Speicherklasse und einer Beanspruchung eines dauerhaften Volumes (Persistent Volume Claim – PVC) dynamisch erstellen.

Dynamische PVC für TKG-Cluster

Zur Ausführung statusbehafteter Arbeitslasten für TKG-Cluster können Sie eine Beanspruchung eines dauerhaften Volumes (Persistent Volume Claim – PVC) erstellen, um dauerhafte Speicherressourcen anzufordern, ohne die Details der zugrunde liegenden Speicherinfrastruktur zu kennen. Der Speicher, der für das PVC verwendet wird, wird aus dem Speicherkontingent für den vSphere-Namespace zugewiesen.

Die Anforderung stellt dynamisch ein dauerhaftes Volume-Objekt und eine übereinstimmende virtuelle Festplatte bereit. Die Beanspruchung ist an das dauerhafte Volume gebunden. Wenn Sie die Beanspruchung löschen, werden das entsprechende dauerhafte Volume-Objekt und die bereitgestellte virtuelle Festplatte ebenfalls gelöscht.

Beim Erstellen der PVC wird das zugrunde liegende dauerhafte Volume dynamisch erstellt. Die PVC verweist auf die Speicherklasse **`tkg-store`**. Die Speicherklasse ist mit dem vSphere-Namespace verknüpft, auf dem der TKG-Zielcluster bereitgestellt wird. Weitere Informationen hierzu finden Sie unter Verwenden von Speicherklassen für persistente Volumes.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: tkg-cluster-pvc
spec:
  accessModes:
    - ReadWriteMany
```

```
  storageClassName: tkg-store
  resources:
    requests:
        storage: 3Gi
```

Erstellen Sie die PVC.

```
kubectl apply -f pvc_name.yaml
```

Überprüfen Sie die PVC.

```
kubectl get pvc my-pvc
```

Geben Sie die PVC im Pod oder in der Bereitstellungsspezifikation an. Beispiel:

```
...
volumes:
    - name: my-pvc
      persistentVolumeClaim:
        claimName: my-pvc
```

Statisches Erstellen persistenter Speicher-Volumes

Sie können ein dauerhaftes Volume (Persistent Volume – PV) in einem TKG 2.0-Cluster mithilfe einer nicht verwendeten Beanspruchung eines dauerhaften Volumes (Persistent Volume Claim, PVC) über den Supervisor statisch erstellen.

Definition eines dauerhaften Volumes

Im Folgenden finden Sie eine Beispieldefinition für ein statisches dauerhaftes Volume (Persistent Volume – PV). Die Definition erfordert eine Speicherklasse und ein Volume-Handle. `volumeHandle` ist der Name einer PVC-Beanspruchung, die auf dem Supervisor im selben vSphere-Namespace erstellt wird, in dem der TKG-Zielcluster bereitgestellt wird. Dieses PVC darf nicht an einen Pod angehängt werden.

Verwenden Sie den folgenden Befehl, um den `storageClassName` abzurufen.

```
kubectl get storageclass
```

Geben Sie für `volumeHandle` den Namen der PVC auf dem Supervisor ein.

```
apiVersion: v1
   kind: PersistentVolume
   metadata:
     name: static-tkg-block-pv
     annotations:
       pv.kubernetes.io/provisioned-by: csi.vsphere.vmware.com
   spec:
     storageClassName: gc-storage-profile
     capacity:
```

```
        storage: 2Gi
      accessModes:
        - ReadWriteOnce
      persistentVolumeReclaimPolicy: Delete
      claimRef:
        namespace: default
        name: static-tkg-block-pvc
      csi:
        driver: "csi.vsphere.vmware.com"
        volumeAttributes:
          type: "vSphere CNS Block Volume"
        volumeHandle: "supervisor-block-pvc-name"  #Enter the PVC name from Supervisor.
```

Verwenden Sie Folgendes, um das PV zu erstellen.

```
kubectl apply -f redis-leader-pvc.yaml -n guestbook
```

Beanspruchung eines persistenten Volumes (PVC) für statisch definiertes PV

Wenn Sie den Supervisor in Zonen bereitgestellt haben.

Legen Sie für `storageClassName` denselben Wert wie im PV fest.

```
kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: static-tkg-block-pvc
   spec:
     accessModes:
       - ReadWriteOnce
     resources:
       requests:
         storage: 2Gi
     storageClassName: gc-storage-profile
     volumeName: static-tkg-block-pv
```

Stellen Sie sicher, dass die PVC an das von Ihnen erstellte PV gebunden ist.

```
kubectl get pv,pvc
```

Bereitstellen der Guestbook-Anwendung in einem TKG-Cluster

Stellen Sie die Guestbook-Anwendung in Ihrem TKG-Cluster bereit und erkunden Sie Kubernetes.

Das Bereitstellen der Guestbook-Anwendung ist eine nützliche Methode zum Erkunden von Kubernetes. Die Anwendung verwendet Bereitstellungs- und ReplicaSet-Objekte, um Pods im Standard-Namespace bereitzustellen und diese Pods mithilfe von Diensten verfügbar zu machen. Guestbook-Daten sind dauerhaft, sodass die Daten beim Ausfall der Anwendung beibehalten werden. Dieses Lernprogramm verwendet die dynamische Beanspruchung eines dauerhaften

Volumes (Persistent Volume Claim, PVC), um dauerhafte Speicherressourcen anzufordern, ohne die Details des zugrunde liegenden Speichers zu kennen. Der Speicher, der für das PVC verwendet wird, wird aus dem Speicherkontingent für den vSphere-Namespace zugewiesen. Container sind standardmäßig flüchtig und statusfrei. Ein gängiger Ansatz bei statusbehafteten Arbeitslasten besteht darin, eine Beanspruchung eines dauerhaften Volumes (Persistent Volume Claim, PVC) zu erstellen. Mit einem PVC können Sie die dauerhaften Volumes mounten und auf den Speicher zugreifen. Die Anforderung stellt dynamisch ein dauerhaftes Volume-Objekt und eine übereinstimmende virtuelle Festplatte bereit. Die Beanspruchung ist an das dauerhafte Volume gebunden. Wenn Sie die Beanspruchung löschen, werden das entsprechende dauerhafte Volume-Objekt und die bereitgestellte virtuelle Festplatte ebenfalls gelöscht.

Voraussetzungen

Informationen hierzu finden Sie unter den folgenden Themen:

- Lernprogramm für die Guestbook-Anwendung in der Kubernetes-Dokumentation
- Stellen Sie einen TKG-Cluster bereit.
- Stellen Sie eine Verbindung mit dem TKG-Cluster her.

Verfahren

1 Melden Sie sich beim TKG-Cluster an.

2 Erstellen Sie den Guestbook-Namespace.

```
kubectl create namespace guestbook
```

Überprüfen Sie:

```
kubectl get ns
```

3 Erstellen Sie rollenbasierte Zugriffssteuerung mithilfe des standardmäßig privilegierten PSP.

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding --
clusterrole=psp:vmware-system-privileged --group=system:authenticated
```

Hinweis Wenn Sie umfassendere Sicherheit benötigen, wenden Sie ein RoleBinding auf den Guestbook-Namespace an.

4 Überprüfen Sie die Speicherklasse oder erstellen Sie eine.

So überprüfen Sie eine vorhandene Speicherklasse:

```
kubectl describe namespace
```

Oder, wenn Sie über vSphere-Administratorberechtigungen verfügen:

```
kubectl get storageclass
```

5 Erstellen Sie die YAML-Dateien für die Beanspruchung eines dauerhaften Volumes (Persistent Volume Claims, PVCs), die die Speicherklasse verwenden.

Verwenden Sie die folgenden YAML-Dateien. Aktualisieren Sie jede mit dem Namen der Speicherklasse.

- Redis-Leader – Beanspruchung eines dauerhaften Volumes
- Redis-Follower – Beanspruchung eines persistenten Volumes

6 Wenden Sie die Guestbook-PVCs auf den Cluster an.

```
kubectl apply -f redis-leader-pvc.yaml -n guestbook
```

```
kubectl apply -f redis-follower-pvc.yaml -n guestbook
```

7 Überprüfen Sie den Status der PVCs.

```
kubectl get pvc,pv -n guestbook
```

Die PVCs und dauerhaften Volumes (persistent volumes, PVs) sind aufgelistet und zur Verwendung verfügbar.

```
NAME                                        STATUS
VOLUME                                      CAPACITY   ACCESS MODES   STORAGECLASS
AGE
persistentvolumeclaim/redis-follower-pvc    Bound    pvc-37b72f35-3de2-4f84-
be7d-50d5dd968f62   2Gi        RWO            tkgs-storage-class   66s
persistentvolumeclaim/redis-leader-pvc      Bound    pvc-2ef51f31-dd4b-4fe2-bf4c-
f0149cb4f3da   2Gi        RWO            tkgs-storage-class   66s

NAME                                                CAPACITY   ACCESS MODES   RECLAIM
POLICY STATUS   CLAIM                 STORAGECLASS
persistentvolume/pvc-2ef51f31-dd4b-4fe2-bf4c   2Gi  RWO            Delete
Bound    guestbook/redis-leader-pvc     tkgs-storage-class
persistentvolume/pvc-37b72f35-3de2-4f84-be7d   2Gi  RWO            Delete
Bound    guestbook/redis-follower-pvc   tkgs-storage-class
```

8 Erstellen Sie die Guestbook-YAML-Dateien.

Verwenden Sie die folgenden YAML-Dateien:

- Redis-Leader-Bereitstellung
- Redis-Follower-Bereitstellung
- Redis Leader Service
- Redis Follower Service
- Guestbook Frontend Deployment
- Guestbook Frontend Service

9 Stellen Sie die Guestbook-Anwendung in ihrem Namespace bereit.

```
kubectl apply -f . --namespace guestbook
```

10 Überprüfen Sie die Erstellung der Guestbook-Ressourcen.

```
kubectl get all -n guestbook
```

```
NAME                                                READY   STATUS
RESTARTS   AGE
pod/guestbook-frontend-deployment-56fc5b6b47-cd58r   1/1     Running
0          65s
pod/guestbook-frontend-deployment-56fc5b6b47-fh6dp   1/1     Running
0          65s
pod/guestbook-frontend-deployment-56fc5b6b47-hgd2b   1/1     Running
0          65s
pod/redis-follower-deployment-6fc9cf5759-99fgw       1/1     Running
0          65s
pod/redis-follower-deployment-6fc9cf5759-rhxf7       1/1     Running
0          65s
pod/redis-leader-deployment-7d89bbdbcf-flt4q         1/1     Running
0          65s

NAME                                 TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)        AGE
service/guestbook-frontend           LoadBalancer   10.10.89.59      10.19.15.99
80:31513/TCP   65s
service/redis-follower               ClusterIP      10.111.163.189   <none>
6379/TCP       65s
service/redis-leader                 ClusterIP      10.111.70.189    <none>
6379/TCP       65s

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/guestbook-frontend-deployment   3/3     3            3           65s
deployment.apps/redis-follower-deployment       1/2     2            1           65s
deployment.apps/redis-leader-deployment         1/1     1            1           65s

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/guestbook-frontend-deployment-56fc5b6b47   3         3         3       65s
replicaset.apps/redis-follower-deployment-6fc9cf5759       2         2         1       65s
replicaset.apps/redis-leader-deployment-7d89bbdbcf         1         1         1       65s
```

11 Rufen Sie die Guestbook-Webseite mit der `External-IP`-Adresse des `service/guestbook-frontend`-Lastausgleichsdiensts auf, in diesem Beispiel die `10.19.15.99`.

Sie sehen die Guestbook-Webschnittstelle und können Werte in die Guestbook-Datenbank eingeben. Wenn Sie die Anwendung neu starten, werden die Daten beibehalten.

YAML für Guestbook-Anwendung

Verwenden Sie die YAML-Beispieldateien, um die Guestbook-Anwendung mit dauerhaftem Datenspeicher bereitzustellen.

Redis-Leader – Beanspruchung eines dauerhaften Volumes

Die Datei `redis-leader-pvc.yaml` ist ein Beispiel für die Beanspruchung eines dauerhaften Volumes, das auf eine benannte Speicherklasse verweist. Um dieses Beispiel zu verwenden, geben Sie den Name der Speicherklasse ein.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: redis-leader-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: tkg-storage-class-name
  resources:
    requests:
      storage: 2Gi
```

Redis-Follower – Beanspruchung eines persistenten Volumes

Die Datei `redis-follower-pvc.yaml` ist ein Beispiel für die Beanspruchung eines dauerhaften Volumes, das auf eine benannte Speicherklasse verweist. Um dieses Beispiel zu verwenden, geben Sie den Name der Speicherklasse ein.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: redis-follower-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: tkg-storage-class-name
  resources:
    requests:
      storage: 2Gi
```

Redis-Leader-Bereitstellung

Die Datei `redis-leader-deployment.yaml` ist ein Beispiel für die Redis-Leader-Bereitstellung mit einem dauerhaften Volume.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-leader-deployment
spec:
```

```
  selector:
    matchLabels:
      app: redis
      role: leader
      tier: backend
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
        role: leader
        tier: backend
    spec:
      containers:
      - name: leader
        image: redis:6.0.5
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
        volumeMounts:
        - name: redis-leader-data
          mountPath: /data
      volumes:
      - name: redis-leader-data
        persistentVolumeClaim:
          claimName: redis-leader-pvc
```

Redis-Follower-Bereitstellung

Die Datei `redis-follower-deployment.yaml` ist ein Beispiel für die Redis-Follower-Bereitstellung mit einem dauerhaften Volume.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-follower-deployment
  labels:
    app: redis
spec:
  selector:
    matchLabels:
      app: redis
      role: follower
      tier: backend
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
        role: follower
        tier: backend
```

```
    spec:
      containers:
      - name: follower
        image: gcr.io/google_samples/gb-redis-follower:v2
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
        ports:
        - containerPort: 6379
        volumeMounts:
        - name: redis-follower-data
          mountPath: /data
      volumes:
      - name: redis-follower-data
        persistentVolumeClaim:
          claimName: redis-follower-pvc
```

Redis Leader Service

Die Datei `redis-leader-service.yaml` ist ein Beispiel für den Redis-Leader-Dienst.

```
apiVersion: v1
kind: Service
metadata:
  name: redis-leader
  labels:
    app: redis
    role: leader
    tier: backend
spec:
  ports:
  - port: 6379
    targetPort: 6379
  selector:
    app: redis
    role: leader
    tier: backend
```

Redis Follower Service

Die Datei `redis-follower-service.yaml` ist ein Beispiel für den Redis-Follower-Dienst.

```
apiVersion: v1
kind: Service
metadata:
  name: redis-follower
  labels:
    app: redis
    role: follower
```

```
    tier: backend
spec:
  ports:
  - port: 6379
  selector:
    app: redis
    role: follower
    tier: backend
```

Guestbook Frontend Deployment

Die Datei `guestbook-frontend-deployment.yaml` ist ein Beispiel für die Guestbook-Frontend-Bereitstellung.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: guestbook-frontend-deployment
spec:
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  replicas: 3
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v5
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
        ports:
        - containerPort: 80
```

Guestbook Frontend Service

Die Datei `guestbook-frontend-service.yaml` ist ein Beispiel für den Guestbook-Frontend-Lastausgleichsdienst.

```
apiVersion: v1
kind: Service
metadata:
  name: guestbook-frontend
  labels:
```

```
    app: guestbook
    tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend
```

Bereitstellen einer StatefulSet-Anwendung über vSphere-Zonen mit einem Volume-Anhang mit später Bindung

Dieses Beispiel zeigt die Bereitstellung einer StatefulSet-Anwendung in einem TKG-Cluster auf Supervisor.

Speicherklasse

Die Speicherklasse gibt es in zwei verschiedenen Editionen. Für diese Bereitstellung verwenden wir die Edition `*-latebinding`.

```
kubectl get sc
NAME                              PROVISIONER              RECLAIMPOLICY
VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
zonal-ds-policy-105               csi.vsphere.vmware.com   Delete
Immediate              true                   17h
zonal-ds-policy-105-latebinding   csi.vsphere.vmware.com   Delete
WaitForFirstConsumer   true                   17h
```

Topologie mit multizonenbasiertem Supervisor

TKG-Cluster wird auf Supervisor über vSphere-Zonen bereitgestellt.

```
kubectl get nodes -L topology.kubernetes.io/zone
NAME                                                          STATUS
ROLES                  AGE   VERSION            ZONE
test-cluster-e2e-script-105-m72sb-2dnsz                       Ready    control-
plane,master   18h   v1.22.6+vmware.1   zone-1
test-cluster-e2e-script-105-m72sb-rmtjn                       Ready    control-
plane,master   18h   v1.22.6+vmware.1   zone-2
test-cluster-e2e-script-105-m72sb-rvhb8                       Ready    control-
plane,master   18h   v1.22.6+vmware.1   zone-3
test-cluster-e2e-script-105-nodepool-1-p86fm-6dfcdc77b7-fxm4s   Ready
<none>                 18h   v1.22.6+vmware.1   zone-1
test-cluster-e2e-script-105-nodepool-2-gx5gs-7cf4895b77-6wlb4   Ready
<none>                 18h   v1.22.6+vmware.1   zone-2
test-cluster-e2e-script-105-nodepool-3-fkkc9-856cd45985-d8nsl   Ready
<none>                 18h   v1.22.6+vmware.1   zone-3
```

StatefulSet

Das StatefulSet (`sts.yaml`) stellt die Anwendung für Pods bereit, von denen jeder über einen persistenten Bezeichner verfügt, der bei jeder Neuplanung beibehalten wird.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  serviceName: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: topology.kubernetes.io/zone
                operator: In
                values:
                - zone-1
                - zone-2
                - zone-3
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - nginx
            topologyKey: topology.kubernetes.io/zone
      containers:
        - name: nginx
          image: gcr.io/google_containers/nginx-slim:0.8
          ports:
            - containerPort: 80
              name: web
          volumeMounts:
            - name: www
              mountPath: /usr/share/nginx/html
            - name: logs
              mountPath: /logs
  volumeClaimTemplates:
    - metadata:
```

```
        name: www
      spec:
        accessModes: [ "ReadWriteOnce" ]
        storageClassName: zonal-ds-policy-105-latebinding
        resources:
          requests:
            storage: 2Gi
    - metadata:
        name: logs
      spec:
        accessModes: [ "ReadWriteOnce" ]
        storageClassName: zonal-ds-policy-105-latebinding
        resources:
          requests:
            storage: 1Gi
```

Bereitstellen der Anwendung

Stellen Sie die Anwendung StatefulSet wie folgt bereit. Bei erfolgreicher Bereitstellung werden Anwendungs-Pods über 3 vSphere-Zonen hinweg geplant, wobei auf den ersten Verbraucher gewartet wird, der die Speicherklasse mit dem Volume mit später Bindung verwendet.

1 Stellen Sie das StatefulSet bereit.

```
kubectl create -f sts.yaml
statefulset.apps/web created
```

2 Überprüfen Sie das StatefulSet.

```
kubectl get statefulset
NAME   READY   AGE
web    3/3     112s
```

3 Überprüfen Sie die Pods.

```
kubectl get pods -o wide
NAME    READY   STATUS    RESTARTS   AGE    IP
NODE                                                          NOMINATED NODE   READINESS
GATES
web-0   1/1     Running   0          117s   172.16.1.2   test-cluster-e2e-script-105-
nodepool-3-fkkc9-856cd45985-d8nsl   <none>           <none>
web-1   1/1     Running   0          90s    172.16.2.2   test-cluster-e2e-script-105-
nodepool-2-gx5gs-7cf4895b77-6wlb4   <none>           <none>
web-2   1/1     Running   0          53s    172.16.3.2   test-cluster-e2e-script-105-
nodepool-1-p86fm-6dfcdc77b7-fxm4s   <none>           <none>
```

4 Überprüfen Sie die zonenübergreifende Pod-Planung und das Volume mit später Bindung.

```
kubectl get pv -o=jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.spec.claimRef.name}
{"\t"}{.spec.nodeAffinity}{"\n"}{end}'
pvc-7010597f-31cf-4ab1-bbd7-98ac04e0c603    www-
web-2    {"required":{"nodeSelectorTerms":[{"matchExpressions":
[{"key":"topology.kubernetes.io/zone","operator":"In","values":["zone-1"]}]}]}}
```

```
pvc-921fadfc-df89-456d-a341-00f4117035f8    logs-
web-0    {"required":{"nodeSelectorTerms":[{"matchExpressions":
[{"key":"topology.kubernetes.io/zone","operator":"In","values":["zone-3"]}]}]}}
pvc-bcb46a24-58cb-4ec7-a964-391fe80400fc    www-
web-1    {"required":{"nodeSelectorTerms":[{"matchExpressions":
[{"key":"topology.kubernetes.io/zone","operator":"In","values":["zone-2"]}]}]}}
pvc-f51a44e5-ec19-4bec-b67a-3e34512049b8    www-
web-0    {"required":{"nodeSelectorTerms":[{"matchExpressions":
[{"key":"topology.kubernetes.io/zone","operator":"In","values":["zone-3"]}]}]}}
pvc-fa68887a-31dd-4d9e-bb39-88653a9d80c9    logs-
web-2    {"required":{"nodeSelectorTerms":[{"matchExpressions":
[{"key":"topology.kubernetes.io/zone","operator":"In","values":["zone-1"]}]}]}}
pvc-fc2cd6f7-b033-48ee-892d-df5318ec6f3e    logs-
web-1    {"required":{"nodeSelectorTerms":[{"matchExpressions":
[{"key":"topology.kubernetes.io/zone","operator":"In","values":["zone-2"]}]}]}}
```

Bereitstellen von AI/ML-Arbeitslasten in TKG-Dienst-Clustern

13

Sie können KI-/ML-Arbeitslasten in TKG-Dienst-Clustern mit der NVIDIA vGPU bereitstellen. Für die Bereitstellung von KI-/ML-Arbeitslasten muss eine Ersteinrichtung durch den vSphere-Administrator und eine bestimmte Konfiguration durch den Cluster-Operator erfolgen. Sobald die Umgebung vGPU-fähig ist, können Entwickler Container-basierte KI-/ML-Arbeitslasten für ihre TKG-Dienst-Cluster bereitstellen.

Lesen Sie als Nächstes die folgenden Themen:

- Informationen zum Bereitstellen von KI-/ML-Arbeitslasten in TKG-Dienst-Clustern
- vSphere-Administrator-Workflow für die Bereitstellung von KI-/ML-Arbeitslasten auf TKGS-Clustern
- Cluster-Operator-Workflow für die Bereitstellung von KI-/ML-Arbeitslasten in TKG-Dienstclustern
- Erstellen einer benutzerdefinierten VM-Klasse für NVIDIA vGPU-Geräte

Informationen zum Bereitstellen von KI-/ML-Arbeitslasten in TKG-Dienst-Clustern

Sie können KI-/ML-Arbeitslasten in TKG-Dienst-Clustern mit der NVIDIA GPU-Technologie bereitstellen.

TKGS-Unterstützung für KI-/ML-Arbeitslasten

Sie können Computing-intensive Arbeitslasten für TKG-Dienst-Cluster bereitstellen. In diesem Kontext handelt es sich bei einer rechenintensiven Arbeitslast um eine KI- (Künstliche Intelligenz) oder ML-Anwendung (Maschinelles Lernen), die die Verwendung eines Geräts für die GPU-Beschleunigung erfordert.

Zur Vereinfachung der Ausführung von KI-/ML-Arbeitslasten in einer Kubernetes-Umgebung ist VMware eine Partnerschaft mit NVIDIA zur Unterstützung der NVIDIA GPU-Cloud-Plattform eingegangen. Dies bedeutet, dass Sie Container-Images aus dem NGC-Katalog in TKGS-Clustern bereitstellen können. Weitere Informationen zur Unterstützung von NVIDIA GPU in vSphere 8 finden Sie im vGPU-Artikel in Tech Zone.

Unterstützte GPU-Modi

Für die Bereitstellung von NVIDIA-basierten KI-/ML-Arbeitslasten in TKG-Dienst-Clustern muss die Ubuntu-Edition von Tanzu Kubernetes Releases (Version 1.22 oder höher) verwendet werden. vSphere unterstützt zwei Modi: NVIDIA Grid vGPU und GPU-Passthrough mithilfe eines Dynamic DirectPath I/O-Geräts. Weitere Informationen finden Sie unter Unterstützte Betriebssysteme und Kubernetes-Plattformen in der NVIDIA-Dokumentation.

Tabelle 13-1. vSphere-VMs mit NVIDIA vGPU

Betriebssystem	TKr	vSphere with Tanzu	Beschreibung
Ubuntu 20.04 LTS	1.22 – 1.2x* (letzte bis einschließlich 1.28)	7.0 U3c 8.0 U2+	Das GPU-Gerät wird vom Treiber des NVIDIA-Hostmanagers virtualisiert, der auf jedem ESXi-Host installiert ist. Das GPU-Gerät wird dann von mehreren virtuellen NVIDIA vGPUs (virtual GPUs) gemeinsam genutzt. **Hinweis** Der vSphere Distributed Resource Scheduler (DRS) verteilt vGPU-VMs weitläufig auf die Hosts, aus denen sich ein vSphere-Cluster zusammensetzt. Weitere Informationen finden Sie unter DRS Placement of vGPU VMs (DRS-Platzierung von vGPU-VMs) im Handbuch zur vSphere-Ressourcenverwaltung. Jede NVIDIA vGPU wird durch die Menge des Arbeitsspeichers auf dem GPU-Gerät definiert. Wenn das GPU-Gerät beispielsweise über Arbeitsspeicher (RAM) von insgesamt 32 GB verfügt, können Sie 8 vGPUs mit jeweils 4 GB Arbeitsspeicher erstellen.

Tabelle 13-2. vSphere-VMs mit GPU-Passthrough

Betriebssystem	TKr	vSphere with Tanzu	Beschreibung
Ubuntu 20.04 LTS	1.22 – 1.2x* (letzte bis einschließlich 1.28)	7.0 U3c 8.0 U2+	In derselben VM-Klasse, in der Sie das NVIDIA vGPU-Profil konfigurieren, integrieren Sie Unterstützung für ein Passthrough-Netzwerkgerät mit Dynamic DirectPath IO. In diesem Fall wird die VM-Platzierung von vSphere DRS festgelegt.

vSphere-Administrator-Workflow für die Bereitstellung von KI-/ML-Arbeitslasten auf TKGS-Clustern

Damit Entwickler KI-/ML-Arbeitslasten auf TKG-Clustern bereitstellen können, richten Sie als vSphere-Administrator die Supervisor-Umgebung zur Unterstützung der NVIDIA GPU-Hardware ein.

Schritt 1 für Administratoren: Überprüfen der Systemanforderungen

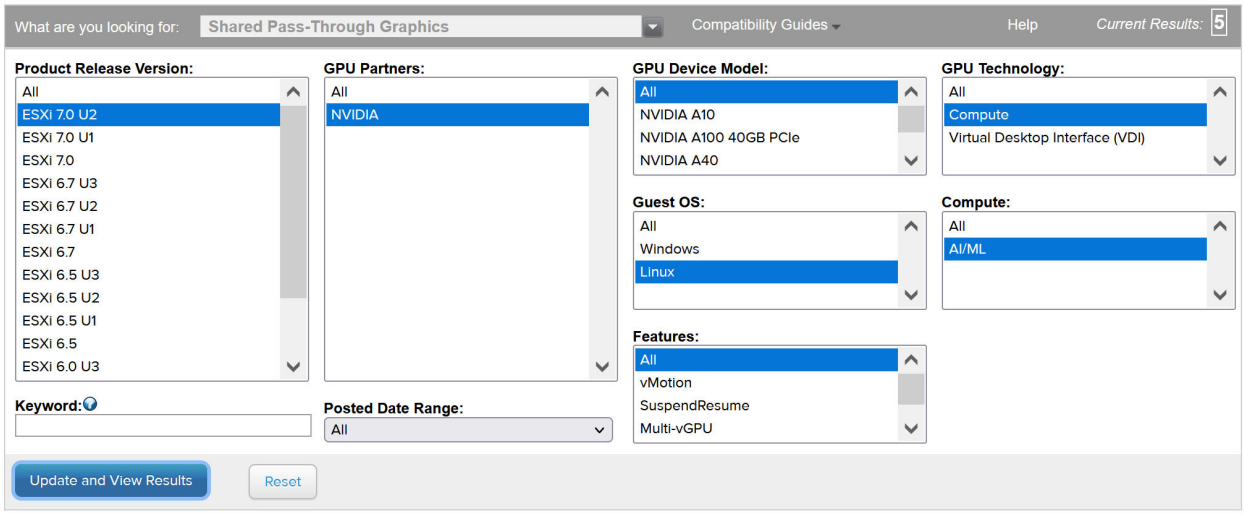
Informationen zum Einrichten der Umgebung für die Bereitstellung von AI/ML-Arbeitslasten auf TKG-Clustern finden Sie in den folgenden Systemanforderungen.

Anforderung	Beschreibung
vSphere 8-Infrastruktur	vCenter Server und ESXi-Hosts
Workload Management-Lizenz	vSphere Namespaces und Supervisor
TKR Ubuntu-OVA	Versionshinweise zu Tanzu Kubernetes-Versionen
NVIDIA vGPU-Hosttreiber	Laden Sie das VIB von der NGC-Website herunter. Weitere Informationen finden Sie in der Dokumentation des vGPU Software-Treibers.
NVIDIA-Lizenzserver für vGPU	Von Ihrer Organisation bereitgestellter FQDN

Schritt 2 für Administratoren: Installieren des unterstützten NVIDIA GPU-Geräts auf ESXi-Hosts

Zur Bereitstellung von KI-/ML-Arbeitslasten auf TKG installieren Sie mindestens ein unterstütztes NVIDIA GPU-Gerät auf allen ESXi-Hosts, aus denen sich der vCenter-Cluster zusammensetzt, auf dem **Arbeitslastverwaltung** aktiviert wird.

Informationen zum Anzeigen kompatibler NVIDIA GPU-Geräte finden Sie im VMware-Kompatibilitätshandbuch.



Click here to **Read** Important Support Information

Click on the 'GPU Device Model' to view more details and to subscribe to RSS feeds.

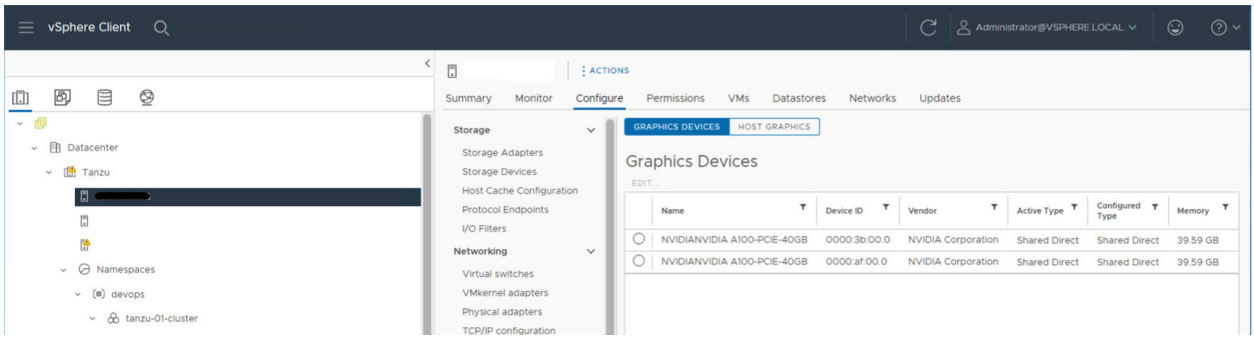
Bookmark | Print | Export to CSV

Search Results: Your search for" Shared Pass-Through Graphics " returned **5 results.** Back to Top Turn Off Auto Scroll Display: 10

GPU Partner	GPU Device Model	ESX Version	Compute
NVIDIA	NVIDIA A10	ESXi 7.0 U2	AI/ML
NVIDIA	NVIDIA A100 40GB PCIe	ESXi 7.0 U2	AI/ML
NVIDIA	NVIDIA A100 80GB PCIe	ESXi 7.0 U2	AI/ML
NVIDIA	NVIDIA A30	ESXi 7.0 U2	AI/ML
NVIDIA	NVIDIA A40	ESXi 7.0 U2	AI/ML

Das NVIDIA GPU-Gerät sollte die aktuellen vGPU-Profile von NVIDIA AI Enterprise (NVAIE) unterstützen. Weitere Informationen finden Sie in der Dokumentation Von der NVIDIA Virtual GPU-Software unterstützte GPUs.

Auf dem folgenden ESXi-Host sind beispielsweise zwei NVIDIA GPU A100-Geräte installiert.



Schritt 3 für Administratoren: Konfigurieren aller ESXi-Hosts für vGPU-Vorgänge

Konfigurieren Sie für jeden ESXi-Host im vCenter-Cluster mit aktivierter **Arbeitslastverwaltung** den Host für NVIDIA vGPU, indem Sie „Direkt freigegeben“ und SR-IOV aktivieren.

Aktivieren von „Direkt freigegeben“ auf allen ESXi-Hosts

Aktivieren Sie zum Entsperren der NVIDIA vGPU-Funktion den Modus **Direkt freigegeben** auf allen ESXi-Hosts, aus denen sich der vCenter-Cluster zusammensetzt, auf dem **Arbeitslastverwaltung** aktiviert ist.

Führen Sie die folgenden Schritte aus, um **Direkt freigegeben** zu aktivieren. Weitere Informationen finden Sie unter Konfigurieren virtueller Grafiken auf vSphere.

1 Melden Sie sich beim vCenter Server mithilfe des vSphere Client an.

2 Wählen Sie einen ESXi-Host im vCenter-Cluster aus.

3 Wählen Sie **Konfigurieren > Hardware > Grafik > Grafikgeräte** aus.

4 Wählen Sie das Gerät für die NVIDIA GPU-Beschleunigung aus.

5 **Bearbeiten** Sie die Einstellungen des Grafikgeräts.

6 Wählen Sie **Direkt freigegeben** aus.

7 Wählen Sie unter **Richtlinie für die Zuweisung von freigegebenen Passthrough-GPUs** für optimale Leistung die Option **VMs über GPUs verteilen** aus.

8 Klicken Sie auf **OK**, um die Konfiguration zu speichern.

9 Beachten Sie, dass die Einstellungen nach dem Neustart des Hosts wirksam werden.

10 Klicken Sie mit der rechten Maustaste auf den ESXi-Host und versetzen Sie ihn in den Wartungsmodus.

11 Starten Sie den Host neu.

12 Wenn der Host erneut ausgeführt wird, beenden Sie den Wartungsmodus.

13 Wiederholen Sie diesen Vorgang für alle ESXi-Hosts in dem vSphere-Cluster, der **Arbeitslastverwaltung** unterstützt.

Einschalten von SR-IOV BIOS für NVIDIA GPU A30- und A100-Geräte

Wenn Sie NVIDIA A30- oder A100-GPU-Geräte verwenden, die für GPU mit mehreren Instanzen (MIG-Modus) benötigt werden, müssen Sie SR-IOV auf dem ESXi-Host aktivieren. Wenn SR-IOV nicht aktiviert ist, können Tanzu Kubernetes-Clusterknoten-VMs nicht gestartet werden. In diesem Fall wird die folgende Fehlermeldung im Bereich **Aktuelle Aufgaben** des vCenter Server angezeigt, für den **Arbeitslastverwaltung** aktiviert ist.

```
Could not initialize plugin libnvidia-vgx.so for vGPU nvidia_aXXX-xx. Failed to start the virtual machine. Module DevicePowerOn power on failed.
```

Melden Sie sich zum Aktivieren von SR-IOV über die Webkonsole beim ESXi-Host an. Wählen Sie **Hardware > verwalten** aus. Wählen Sie das NVIDIA GPU-Gerät aus und klicken Sie auf **SR-IOV konfigurieren**. An dieser Stelle können Sie SR-IOV aktivieren. Weitere Informationen finden Sie unter Single Root I/O Virtualization (SR-IOV) in der vSphere-Dokumentation.

vGPU mit Dynamic DirectPath IO (Passthrough-fähiges Gerät)

Wenn Sie vGPU mit Dynamic DirectPath IO verwenden, führen Sie die folgende zusätzliche Konfiguration durch.

1 Melden Sie sich über vSphere Client bei vCenter Server an.

2 Wählen Sie den ESXi-Zielhost im vCenter-Cluster aus.

3 Wählen Sie **Konfigurieren > Hardware > PCI-Geräte** aus.

4 Wählen Sie die Registerkarte **Alle PCI-Geräte** aus.

5 Wählen Sie das Zielgerät für die NVIDIA GPU-Beschleunigung aus.

6 Klicken Sie auf **Passthrough umschalten**.

7 Klicken Sie mit der rechten Maustaste auf den ESXi-Host und versetzen Sie ihn in den Wartungsmodus.

8 Starten Sie den Host neu.

9 Wenn der Host erneut ausgeführt wird, beenden Sie den Wartungsmodus.

Schritt 4 für Administratoren: Installieren des Treibers für den NVIDIA-Hostmanager auf allen ESXi-Hosts

Zum Ausführen von Tanzu Kubernetes-Clusterknoten-VMs mit der NVIDIA vGPU-Grafikbeschleunigung installieren Sie den Treiber des NVIDIA-Hostmanagers auf allen ESXi-Hosts, aus denen sich der vCenter-Cluster zusammensetzt, in dem **Arbeitslastverwaltung** aktiviert wird.

Die Treiberkomponenten des NVIDIA vGPU-Hostmanagers sind in einem vSphere-Installationspaket (VIB) enthalten. Das NVAIE-VIB wird Ihnen von Ihrer Organisation über das NVIDIA GRID-Lizenzierungsprogramm zur Verfügung gestellt. VMware stellt NVAIE-VIBs weder bereit noch können diese heruntergeladen werden. Im Rahmen des NVIDIA-Lizenzierungsprogramms richtet Ihre Organisation einen Lizenzierungsserver ein. Weitere Informationen finden Sie in der Kurzanleitung der Virtual GPU-Software von NVIDIA.

Sobald die NVIDIA-Umgebung eingerichtet ist, führen Sie den folgenden Befehl auf allen ESXi-Hosts aus und ersetzen Sie die Adresse des NVIDIA-Lizenzservers und die NVAIE VIB-Version durch die entsprechenden Werte für Ihre Umgebung. Weitere Informationen finden Sie unter Installieren und Konfigurieren des NVIDIA-VIB auf ESXi in der Knowledgebase von VMware Support.

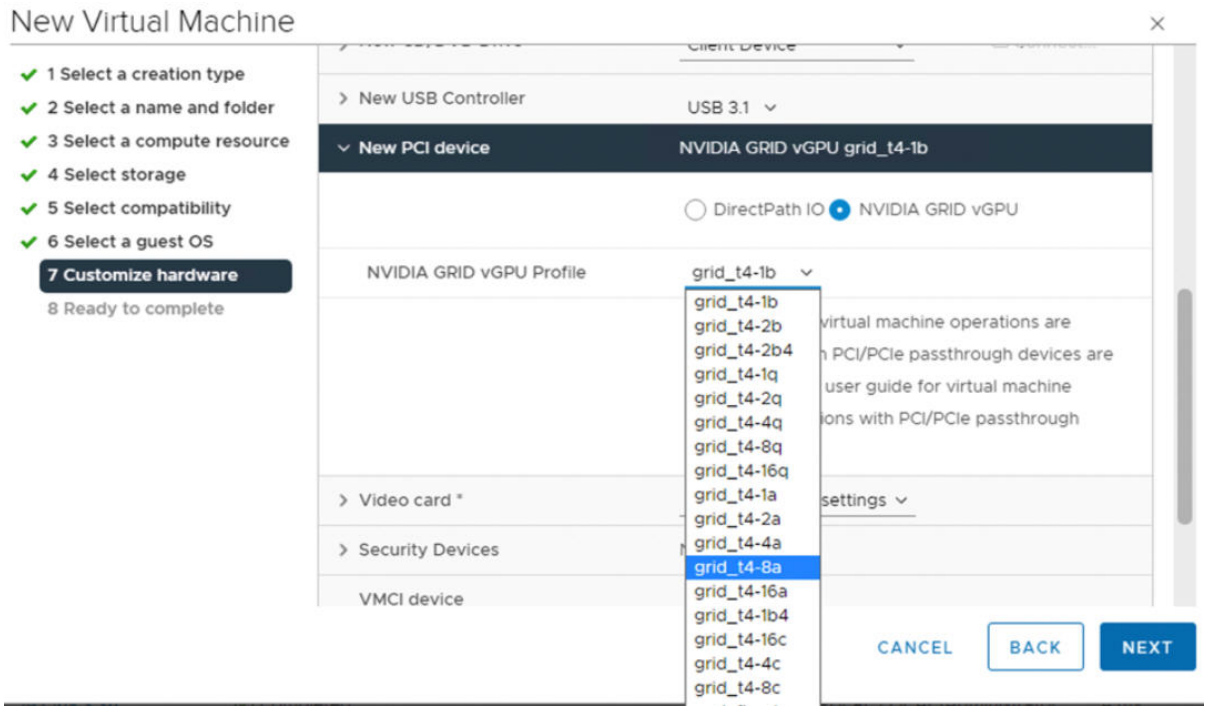
Hinweis Die auf ESXi-Hosts installierte Version des NVAIE-VIB muss mit der auf den Knoten-VMs installierten vGPU-Softwareversion übereinstimmen. Bei der folgenden Version handelt es sich lediglich um ein Beispiel.

```
esxcli system maintenanceMode set --enable true
esxcli software vib install -v ftp://server.domain.example.com/nvidia/signed/
NVIDIA_bootbank_NVIDIA-VMware_ESXi_7.0_Host_Driver_460.73.02-1OEM.700.0.0.15525992.vib
esxcli system maintenanceMode set --enable false
/etc/init.d/xorg restart
```

Schritt 5 für Administratoren: Sicherstellen, dass ESXi-Hosts für NVIDIA vGPU-Vorgänge zur Verfügung stehen

Um sicherzustellen, dass alle ESXi-Hosts für NVIDIA vGPU-Vorgänge zur Verfügung stehen, führen Sie die folgenden Prüfungen auf allen ESXi-Hosts im vCenter-Cluster durch, in dem **Arbeitslastverwaltung** aktiviert ist:

- Melden Sie sich per SSH beim ESXi-Host an, wechseln Sie in den Shell-Modus und führen Sie den Befehl `nvidia-smi` aus. Bei der NVIDIA-Systemverwaltungsschnittstelle handelt es sich um ein Befehlszeilendienstprogramm, das vom NVIDIA vGPU-Hostmanager bereitgestellt wird. Wenn Sie diesen Befehl ausführen, werden die GPUs und Treiber auf dem Host zurückgegeben.
- Führen Sie den folgenden Befehl aus, um sicherzustellen, dass der NVIDIA-Treiber ordnungsgemäß installiert ist: `esxcli software vib list | grep NVIDA`.
- Stellen Sie sicher, dass der Host mit „Direkt freigegeben“ für GPU konfiguriert und SR-IOV eingeschaltet ist (bei Verwendung von NVIDIA A30- oder A100-Geräten).
- Erstellen Sie mithilfe des vSphere Clients auf dem für GPU konfigurierten ESXi-Host eine neue virtuelle Maschine mit einem im Lieferumfang enthaltenen PCI-Gerät. Das NVIDIA vGPU-Profil sollte angezeigt werden und auswählbar sein.



Schritt 6 für Administratoren: Aktivieren der Arbeitslastverwaltung

Informationen zum Aktivieren der **Arbeitslastverwaltung** finden Sie unter Bereitstellen von TKG-Dienst-Clustern.

Hinweis Überspringen Sie diesen Schritt, wenn Sie bereits über einen vSphere-Cluster mit aktivierter **Arbeitslastverwaltung** verfügen. Als Voraussetzung hierfür gilt, dass der Cluster die für vGPU konfigurierten ESXi-Hosts verwendet.

Schritt 7 für Administratoren: Erstellen oder Aktualisieren einer Inhaltsbibliothek mit einer TKR Ubuntu-Version

Das Ubuntu-Betriebssystem wird für NVIDIA vGPU benötigt. Sie können die PhotonOS-Edition einer Tanzu Kubernetes-Version nicht für vGPU-Cluster verwenden.

VMware stellt Ubuntu-Editionen von Tanzu Kubernetes-Versionen bereit. Ab vSphere 8 wird die Ubuntu-Edition mithilfe einer Anmerkung in der Cluster-YAML angegeben.

Erstellen oder aktualisieren Sie eine vorhandene Inhaltsbibliothek mit einer unterstützten Ubuntu TKR. Weitere Informationen finden Sie unter Kapitel 5 Verwalten von Kubernetes-Versionen für TKG-Dienst-Cluster.

Hinweis Überspringen Sie diesen Schritt, wenn Sie bereits eine vorhandene TKR-Inhaltsbibliothek auf vCenter konfiguriert haben. Erstellen Sie keine zweite Inhaltsbibliothek für TKRs. Dies kann zu einer Instabilität des Systems führen.

Schritt 8 für Administratoren: Erstellen einer benutzerdefinierten VM-Klasse mit dem vGPU-Profil

Erstellen Sie eine benutzerdefinierte VM-Klasse mit einem vGPU-Profil. Sie verwenden diese VM-Klasse dann in der Clusterspezifikation, um die TKGS-Clusterknoten zu erstellen. Weitere Informationen finden Sie unter: Erstellen einer benutzerdefinierten VM-Klasse für NVIDIA vGPU-Geräte.

Schritt 9 für Administratoren: Konfigurieren des vSphere-Namespace

Erstellen Sie einen vSphere-Namespace für jeden bereitzustellenden TKG vGPU-Cluster. Weitere Informationen hierzu finden Sie unter Erstellen eines vSphere-Namespace für das Hosting von TKG-Dienst-Clustern.

Konfigurieren Sie den vSphere-Namespace, indem Sie vSphere SSO-Benutzer/Gruppen mit Bearbeitungsberechtigungen hinzufügen und eine Speicherrichtlinie für dauerhafte Volumes anhängen. Weitere Informationen hierzu finden Sie unter Konfigurieren eines vSphere-Namespace für TKG-Dienst-Cluster.

Verknüpfen Sie die TKR-Inhaltsbibliothek, in der das gewünschte Ubuntu-Image gespeichert ist, mit dem vSphere-Namespace. Weitere Informationen hierzu finden Sie unter Verknüpfen der TKR-Inhaltsbibliothek mit dem TKG-Dienst.

Verknüpfen Sie die benutzerdefinierte VM-Klasse mit dem vSphere-Namespace.

- Wählen Sie unter „vSphere-Namespace auswählen“ die Kachel **VM-Dienst** aus und klicken Sie auf **VM-Klassen verwalten**.
- Suchen Sie in der Liste der Klassen nach der von Ihnen erstellten benutzerdefinierten VM-Klasse.
- Aktivieren Sie die Klasse und klicken Sie auf **Hinzufügen**.

Weitere Anleitungen finden Sie unter Verknüpfen von VM-Klassen mit dem vSphere-Namespace.

Schritt 10 für Administratoren: Überprüfen der Bereitschaft von Supervisor

Mithilfe der letzten Verwaltungsaufgabe wird sichergestellt, dass Supervisor bereitgestellt wird und vom Cluster-Operator zur Bereitstellung eines TKG-Clusters für KI-/ML-Arbeitslasten verwendet werden kann.

Weitere Informationen finden Sie unter Herstellen einer Verbindung zu TKG-Dienst-Clustern mithilfe der vCenter SSO-Authentifizierung.

Cluster-Operator-Workflow für die Bereitstellung von KI-/ML-Arbeitslasten in TKG-Dienstclustern

Damit Entwickler KI-/ML-Arbeitslasten auf TKG-Dienstclustern bereitstellen können, erstellen Sie als Cluster-Operator einen oder mehrere Kubernetes-Cluster und installieren Sie die NVIDIA-Netzwerk- und GPU-Operatoren auf diesen Clustern.

Schritt 1 für Operatoren: Überprüfen der Voraussetzungen

Bei diesen Anweisungen wird davon ausgegangen, dass der vSphere-Administrator die Umgebung für NVIDIA GPU eingerichtet hat. Weitere Informationen finden Sie unter vSphere-Administrator-Workflow für die Bereitstellung von KI-/ML-Arbeitslasten auf TKGS-Clustern.

Bei diesen Anweisungen wird davon ausgegangen, dass Sie die NVIDIA AI Enterprise (NVAIE)-Edition des GPU-Operators installieren, der für die Verwendung mit vSphere IaaS control plane vorkonfiguriert und optimiert ist. Der NVAIE GPU Operator unterscheidet sich vom GPU-Operator, der im öffentlichen NGC-Katalog verfügbar ist. Weitere Informationen finden Sie unter NVIDIA AI Enterprise.

Bei diesen Anweisungen wird davon ausgegangen, dass Sie eine Version des NVAIE GPU-Operators und des vGPU-Treibers verwenden, die über ein passendes VIB für ESXi verfügt. Weitere Informationen finden Sie unter NVIDIA GPU Operator-Versionsverwaltung.

Bei der Bereitstellung des TKG-Clusters müssen Sie die Ubuntu-Edition des TKR verwenden. Mit TKG auf vSphere 8 Supervisor wird die Ubuntu-Edition in der Cluster-YAML über eine Anmerkung angegeben.

Schritt 2 für Operatoren: Bereitstellen eines TKGS-Clusters für NVIDIA vGPU

VMware bietet native TKGS-Unterstützung für virtuelle NVIDIA GPUs auf NVIDIA GPU-zertifizierten Servern mit NVIDIA GPU-Operator und NVIDIA-Netzwerkoperator. Sie installieren diese Operatoren in einem TKGS-Arbeitslastcluster. Führen Sie die folgenden Schritte aus, um einen TKGS-Cluster zum Hosten von vGPU-Arbeitslasten bereitzustellen.

1 Installieren Sie den Kubernetes-CLI-Tools für vSphere.

   Weitere Informationen finden Sie unter Installieren des Kubernetes-CLI-Tools für vSphere.

2 Authentifizieren Sie sich mithilfe des vSphere-Plug-In für kubectl beim Supervisor.

```
kubectl vsphere login --server=IP-ADDRESS-or-FQDN --vsphere-username USERNAME
```

   Hinweis Der FQDN kann nur verwendet werden, wenn Supervisor damit aktiviert ist.

3 Führen Sie mithilfe von kubectl einen Kontextwechsel zum vSphere-Namespace durch, den der vSphere-Administrator für den TKGS vGPU-Cluster erstellt hat.

```
kubectl config get-contexts
```

```
kubectl config use-context TKG-GPU-CLUSTER-NAMESPACE
```

4 Rufen Sie den Namen der benutzerdefinierten VM-Klasse mit dem vGPU-Profil ab, das der vSphere-Administrator erstellt hat.

```
kubectl get virtualmachineclass
```

   Hinweis Die VM-Klasse muss an den Ziel-vSphere-Namespace gebunden werden.

5 Rufen Sie TKR NAME für die Ubuntu-Tanzu Kubernetes-Version ab, die der vSphere-Administrator über die Inhaltsbibliothek synchronisiert und dem vSphere-Namespace hinzugefügt hat.

```
kubectl get tkr
```

6 Erstellen Sie die YAML für die Bereitstellung des vGPU-fähigen TKG-Clusters.

   a Geben Sie an, welche Bereitstellungs-API für TKGS-Cluster verwendet werden soll: v1alpha3-API oder v1beta1-API: APIs für die TKG-Cluster-Bereitstellung.

   b Abhängig von der ausgewählten API finden Sie entsprechende Informationen im Ubuntu-Beispiel für diese API.

      - v1alpha3-Beispiel: TKC mit Ubuntu-TKR

- v1beta1-Beispiel: Cluster mit Ubuntu-TKR

Hinweis Sie müssen ein Ubuntu-Betriebssystem-Image verwenden. Photon OS kann nicht verwendet werden.

c Verwenden Sie die Informationen, die Sie aus der Ausgabe der vorhergehenden Befehle abgerufen haben, um die Spezifikation des TKGS-Clusters anzupassen.

7 Stellen Sie den Cluster bereit, indem Sie den folgenden kubectl-Befehl ausführen.

```
kubectl apply -f CLUSTER-NAME.yaml
```

Beispiel:

```
kubectl apply -f tkg-gpu-cluster-1.yaml
```

8 Überprüfen Sie die Clusterbereitstellung.

Überwachen Sie die Bereitstellung von Clusterknoten mithilfe von kubectl.

```
kubectl get tanzukubernetesclusters -n NAMESPACE
```

9 Melden Sie sich beim TKGS vGPU-Cluster mithilfe des vSphere-Plug-In für kubectl an.

```
kubectl vsphere login --server=IP-ADDRESS-or-FQDN --vsphere-username USERNAME \
--tanzu-kubernetes-cluster-name CLUSTER-NAME --tanzu-kubernetes-cluster-namespace
NAMESPACE-NAME
```

10 Überprüfen Sie den Cluster.

Verwenden Sie die folgenden Befehle zum Überprüfen des Clusters:

```
kubectl cluster-info
```

```
kubectl get nodes
```

```
kubectl get namespaces
```

```
kubectl api-resources
```

Schritt 3 für Operatoren: Installieren des NVIDIA-Netzwerkoperators

Der NVIDIA-Netzwerkoperator nutzt benutzerdefinierte Kubernetes-Ressourcen und das Operator-Framework, um das Netzwerk für vGPU zu optimieren. Weitere Informationen finden Sie unter NVIDIA-Netzwerkoperator.

1 Stellen Sie sicher, dass Sie beim TKGS vGPU-Arbeitslastcluster angemeldet sind und dass der Kontext auf den Namespace des TKGS vGPU-Arbeitslastclusters festgelegt ist.

Beachten Sie gegebenenfalls die Anweisungen unter Schritt 2 für Operatoren: Bereitstellen eines TKGS-Clusters für NVIDIA vGPU.

2 Installieren Sie Helm und beachten Sie dabei die Helm-Dokumentation.

3 Rufen Sie das Helm-Diagramm des NVIDIA-Netzwerkoperators ab.

```
helm fetch https://helm.ngc.nvidia.com/nvaie/charts/network-operator-v1.1.0.tgz --
username='$oauthtoken' --password=<YOUR API KEY> --untar
```

4 Erstellen Sie eine YAML-Datei für die Konfigurationswerte.

```
vi values.yaml
```

5 Befüllen Sie die Datei `values.yaml` mit den folgenden Informationen.

```
deployCR: true
 ofedDriver:
  deploy: true
 rdmaSharedDevicePlugin:
  deploy: true
  resources:
   - name: rdma_shared_device_a
   vendors: [15b3]
   devices: [ens192]
```

6 Installieren Sie den NVIDIA-Netzwerkoperator mithilfe des folgenden Befehls.

```
helm install network-operator -f ./values.yaml -n network-operator --create-namespace --
wait network-operator/
```

Schritt 4 für Operatoren: Installieren des NVIDIA GPU-Operators

NVIDIA stellt Kunden einen vorkonfigurierten GPU-Operator für NVIDIA AI Enterprise zur Verfügung. Bei diesen Anweisungen wird davon ausgegangen, dass Sie diese vorkonfigurierte Version des GPU-Operators verwenden. Diese Anweisungen basieren auf den Anweisungen von NVIDIA zum Installieren des GPU-Operators, wurden aber für TKG auf vSphere 8 aktualisiert.

Führen Sie die folgenden Schritte aus, um den GPU-Operator NVIDIA AI Enterprise in dem von Ihnen bereitgestellten TKG-Cluster zu installieren.

1 Stellen Sie sicher, dass Sie beim TKGS vGPU-Arbeitslastcluster angemeldet sind und dass der Kontext auf den Namespace des TKGS vGPU-Arbeitslastclusters festgelegt ist.

   Beachten Sie gegebenenfalls die Anweisungen unter Schritt 2 für Operatoren: Bereitstellen eines TKGS-Clusters für NVIDIA vGPU.

2 Installieren Sie Helm gegebenenfalls unter Verwendung der Helm-Dokumentation.

3 Erstellen Sie den Kubernetes-Namespace `gpu-operator`.

```
kubectl create namespace gpu-operator
```

4 Erstellen Sie eine leere vGPU-Lizenzkonfigurationsdatei.

```
sudo touch gridd.conf
```

5 Generieren Sie ein NLS-Clientlizenztoken und laden Sie es herunter.

Weitere Informationen finden Sie in *Abschnitt 4.6. Generieren eines Clientkonfigurationstokens* des Benutzerhandbuchs zum NVIDIA-Lizenzsystem.

6 Benennen Sie das NLS-Clientlizenztoken um, das Sie nach `client_configuration_token.tok` heruntergeladen haben.

7 Erstellen Sie das ConfigMap-Objekt `licensing-config` im Namespace `gpu-operator`.

Berücksichtigen Sie die Konfigurationsdatei für die vGPU-Lizenz (`gridd.conf`) und das NLS-Clientlizenztoken (`*.tok`) in dieser ConfigMap.

```
kubectl create configmap licensing-config \
    -n gpu-operator --from-file=gridd.conf --from-file=<path>/
client_configuration_token.tok
```

8 Erstellen Sie einen geheimen Schlüssel zum Abrufen von Images für die private Registrierung, die den containerisierten NVIDIA vGPU-Softwaregrafiktreiber für Linux zur Verwendung mit NVIDIA GPU Operator enthält.

Erstellen Sie einen geheimen Schlüssel zum Abrufen von Images im Namespace `gpu-operator` mit dem Namen des Registrierungsgeheimschlüssels `ngc-secret` und dem Namen der privaten Registrierung `nvcr.io/nvaie`. Schließen Sie den NGC-API-Schlüssel und die E-Mail-Adresse in die angegebenen Felder ein.

```
kubectl create secret docker-registry ngc-secret \
--docker-server='nvcr.io/nvaie' \
--docker-username='$oauthtoken' \
--docker-password=<YOUR_NGC_API_KEY> \
--docker-email=<YOUR_EMAIL_ADDRESS> \
-n gpu-operator
```

9 Laden Sie das Helm-Diagramm für NVAIE GPU Operator, Version 2.2 herunter.

Ersetzen Sie „IHR API-SCHLÜSSEL“.

```
helm fetchhttps://helm.ngc.nvidia.com/nvaie/charts/gpu-operator-2-2-v1.11.1.tgz--username='
$oauthtoken' \
--password=<YOUR API KEY>
```

10 Installieren Sie NVAIE GPU Operator, Version 2.2 im TKG-Cluster.

```
helm install gpu-operator ./gpu-operator-2-2-v1.11.1.tgz -n gpu-operator
```

Schritt 5 für Operatoren: Bereitstellen einer KI-/ML-Arbeitslast

Der NVIDIA GPU-Cloud-Katalog enthält mehrere standardmäßige Container-Images, die Sie zum Ausführen von KI-/ML-Arbeitslasten auf Ihren vGPU-fähigen Tanzu Kubernetes-Clustern verwenden können. Weitere Informationen zu den verfügbaren Images finden Sie in der Dokumentation zu NGC.

Erstellen einer benutzerdefinierten VM-Klasse für NVIDIA vGPU-Geräte

In diesem Thema finden Sie Informationen zum Erstellen einer benutzerdefinierten VM-Klasse für NVIDIA GRID vGPU-Geräte.

Erstellen einer benutzerdefinierten VM-Klasse mit vGPU-Profil (v8 U2 P03 und höher)

Mithilfe von NVIDIA vGPU (Virtual Graphics Processing Unit) können mehrere virtuelle Maschinen (VMs) eine einzelne physische GPU gemeinsam nutzen. Zur Verwendung von vGPUs mit TKGS-Clustern definieren Sie eine benutzerdefinierte VM-Klasse. Ab dieser Version steht ein neuer Assistent zum Definieren benutzerdefinierter VM-Klassen zur Verfügung. Anders als bei der Erstellen einer benutzerdefinierten VM-Klasse mit einem vGPU-Profil (v8 U2 und früher) zum Definieren einer benutzerdefinierten VM-Klasse wird das vGPU-Profil aus dem Gerät gelesen und nicht in der VM-Klasse konfiguriert.

Der VM-Operator führt eine Abfrage in der vCenter-Bestandsliste durch, um alle vGPU-Geräte abzurufen, die auf ESXi-Hosts installiert sind, die den vSphere-Cluster bilden, in dem Supervisor bereitgestellt wird. Das zugehörige Profil wird vom vGPU-Gerät definiert. Der Name des vGPU-Geräts gibt an, ob es sich bei dem Profil um eine GPU mit mehreren Instanzen (MIG) oder um GPU mit Timesharing handelt. „MIG“ teilt die Berechnung auf und ermöglicht die parallele Ausführung mehrerer Workloads auf einer einzelnen GPU. „Timesharing“ bietet gemeinsamen Zugriff auf die GPU. Der MIG-Modus basiert auf einer neueren GPU-Architektur und wird nur auf NVIDIA A100- und A30-Geräten unterstützt. Weitere Informationen finden Sie in der NVIDIA-Dokumentation.

Beispiel: Das GPU-Gerät „grid-a100-40c“ stellt ein vGPU-Profil mit Timesharing bereit, das einer VM ein NVIDIA A100 GPU-Gerät mit 40 GB Arbeitsspeicher zuweist. Das entsprechende MIG-basierte vGPU-Profil wäre das Gerät „grid-a100-7-40c“. An der zusätzlichen Zahl zwischen dem Gerät und dem RAM können Sie sehen, dass es sich um ein MIG-Profil handelt. Mit der Zahl „7“ wird angegeben, dass auf dem GPU-Gerät sieben Berechnungsstränge ausgeführt werden. MIG-basierte vGPU-Profile können 1, 2, 3 oder 7 Berechnungsstränge aufweisen.

1 Wählen Sie im vSphere Client-Startmenü die Option **Arbeitslastverwaltung > Dienste** aus.

2 Wählen Sie die Registerkarte **VM-Klassen** aus.

3 Klicken Sie auf **VM-Klasse erstellen**.

   Mit dieser Aktion wird der Assistent zum Erstellen von VM-Klassen gestartet, der Sie beim Erstellen einer VM-Klasse unterstützt.

4 Geben Sie unter **Name** einen Namen für die VM-Klasse ein und klicken Sie auf **Weiter**.

Mit dem Namen der VM-Klasse wird die VM-Klasse angegeben. Geben Sie einen eindeutigen DNS-konformen Namen ein, der diesen Anforderungen entspricht:

- Verwenden Sie einen eindeutigen Namen, der die Namen der standardmäßigen oder benutzerdefinierten VM-Klassen in Ihrer Umgebung nicht dupliziert.
- Verwenden Sie eine alphanumerische Zeichenfolge mit einer maximalen Länge von 63 Zeichen.
- Verwenden Sie keine Großbuchstaben oder Leerzeichen.
- Verwenden Sie einen Bindestrich an einer beliebigen Stelle außer als erstes oder letztes Zeichen. Beispiel: `vm-class1`.
- Nachdem Sie die VM-Klasse erstellt haben, können Sie ihren Namen nicht mehr ändern.

5 Wählen Sie für **Kompatibilität** die Option **ESXi 8.0 U2 und höher** aus und klicken Sie auf **Weiter**.

Weitere Informationen finden Sie unter Kompatibilität der virtuellen Maschine.

Hinweis Nach der Erstellung einer VM-Klasse können Sie deren Hardwarekompatibilität nicht mehr ändern.

6 Fügen Sie unter **Konfiguration > Virtuelle Hardware** das NVIDIA GPU-Gerät zur VM-Klasse hinzu.

a Wählen Sie **Konfiguration > Virtuelle Hardware > Neues Gerät hinzufügen > PCI-Gerät** aus.

b Wählen Sie das gewünschte NVIDIA GRID vGPU-Gerät in der Liste aus. Zwei Arten von NVIDIA GRID vGPU-Profilen stehen zur Verfügung: **Timesharing** und **GPU-Freigabe mit mehreren Instanzen**. Das Profil wird vom System erkannt, wenn Sie das Gerät auswählen.

Hinweis Sie können einer VM-Klasse nur ein NVIDIA GRID vGPU-Gerät vom Typ „MIG-Profil“ hinzufügen.

c Klicken Sie auf **Auswählen**, und das **Neue PCI-Gerät** wird auf der Registerkarte „Virtuelle Hardware“ angezeigt.

7 Geben Sie unter **Konfiguration > Virtuelle Hardware** die gewünschten Einstellungen für **CPU**, **Arbeitsspeicher**, **Neues PCI-Gerät**, **Grafikkarte** und **Sicherheitsgeräte** an.

Tabelle 13-3. CPU-Konfiguration

Einstellung	Konfiguration
CPU	Wählen Sie die Anzahl virtueller CPUs für die VM aus. Weitere Informationen finden Sie unter Konfiguration und Einschränkungen der virtuellen CPU.
CPU-Topologie	Beim Einschalten zugewiesen

Tabelle 13-3. CPU-Konfiguration (Fortsetzung)

Einstellung	Konfiguration
Reservierung	Die Reservierung muss zwischen 0 und 10 MHz liegen.
Grenzwert	Der Grenzwert muss größer oder gleich 10 MHz sein.
Anteile	Zu den Optionen gehören „Niedrig“, „Normal“, „Hoch“ und „Benutzerdefiniert“.
Hardwarevirtualisierung	Wählen Sie diese Option aus, um hardwaregestützte Virtualisierung für das Gastbetriebssystem bereitzustellen.
Leistungsindikatoren	Leistungsindikatoren für virtualisierte CPU aktivieren
Affinität wird geplant	Wählen Sie die Affinität eines physischen Prozessors für diese virtuelle Maschine aus. Verwenden Sie '-' für Bereiche und ',', um Werte zu trennen. Beispielsweise gibt „0, 2, 4-7“ die Prozessoren 0, 2, 4, 5, 6 und 7 an. Löschen Sie die Zeichenfolge, um die Affinitätseinstellungen zu entfernen.
I/O MMU	Wählen Sie diese Option aus, um MMU (Memory Management Unit) (Seite zu Festplatte) zu aktivieren.

Tabelle 13-4. Arbeitsspeicherkonfiguration

Einstellung	Konfiguration
Arbeitsspeicher	Wählen Sie die Größe des Arbeitsspeichers für die VM aus. Weitere Informationen finden Sie unter Maximaler Arbeitsspeicher der virtuellen Maschine.
Reservierung	Geben Sie die garantierte Mindestzuteilung für eine virtuelle Maschine an oder reservieren Sie den gesamten Gastarbeitsspeicher. Wenn die Reservierung nicht eingehalten werden kann, wird die virtuelle Maschine nicht ausgeführt.
Grenzwert	Wählen Sie die zu begrenzende Arbeitsspeichermenge aus, um einen Grenzwert für den Verbrauch an Arbeitsspeicher für eine virtuelle Maschine festzulegen.
Anteile	Wählen Sie die Menge des gemeinsam zu nutzenden Arbeitsspeichers aus. Anteile stellen eine relative Metrik zum Zuteilen von Arbeitsspeicherkapazität dar. Weitere Informationen finden Sie unter Gemeinsame Arbeitsspeichernutzung.
Arbeitsspeicher-Hotplug	Aktivieren Sie diese Option, damit Arbeitsspeicherressourcen zu einer eingeschalteten VM hinzugefügt werden können. Weitere Informationen finden Sie unter Einstellungen zum Hinzufügen von Arbeitsspeicher im laufenden Betrieb .

Tabelle 13-5. Neues PCI-Gerät > GPU-Freigabekonfiguration

Modus „Timesharing“	MIG-Modus
Im Modus „Timesharing“ weist der vGPU-Scheduler die GPU an, die Arbeit für jede vGPU-fähige VM fortlaufend über einen längeren Zeitraum durchzuführen, um die Leistung bestmöglich auf die vGPUs zu verteilen.	Im MIG-Modus können mehrere vGPU-fähige VMs gleichzeitig auf einem einzelnen GPU-Gerät ausgeführt werden. Wenn die MIG-Option nicht angezeigt wird, wird sie vom ausgewählten PCI-Gerät nicht unterstützt.

Tabelle 13-6. Konfigurieren einer Grafikkarte

Einstellung	Konfiguration
Grafikkarte	Geben Sie an, ob Einstellungen von der Hardware automatisch erkannt werden sollen, oder geben Sie benutzerdefinierte Einstellungen ein. Bei Auswahl von „Automatische Erkennung“ sind andere Einstellungen nicht konfigurierbar.
Anzahl der Anzeigen	Wählen Sie die Anzahl der Anzeigen aus.
Gesamter Videoarbeitsspeicher	Geben Sie den gesamten Videoarbeitsspeicher in MB ein.
3D-Grafik	Wählen Sie diese Option aus, um 3D-Unterstützung zu aktivieren.

Tabelle 13-7. Konfigurieren der Sicherheitsgeräte

„Einstellungen“	Konfiguration
Sicherheitsgerät	Wenn das SGX-Sicherheitsgerät installiert ist, können Sie die VM-Einstellungen an dieser Stelle konfigurieren. Andernfalls ist dieses Feld nicht konfigurierbar. Weitere Informationen finden Sie in der SGX-Dokumentation.

8 Wählen Sie die Registerkarte **Konfiguration > VM-Optionen** aus und konfigurieren Sie alle zusätzlichen VM-Einstellungen. Weitere Informationen finden Sie unter Konfigurieren der Optionen der virtuellen Maschine.

9 Wählen Sie die Registerkarte **Konfiguration > Erweiterte Parameter** aus und fügen Sie alle Attribute für die VM-Klasse hinzu.

10 Klicken Sie auf **Weiter**.

11 Überprüfen Sie auf der Seite **Überprüfen und bestätigen** die Details und klicken Sie auf **Beenden**.

12 Verknüpfen Sie die neue VM-Klasse mit dem vSphere-Namespace. Weitere Informationen hierzu finden Sie unter Verknüpfen von VM-Klassen mit dem vSphere-Namespace.

Abbildung 13-1. NVIDIA vGPU – Geräteauswahl

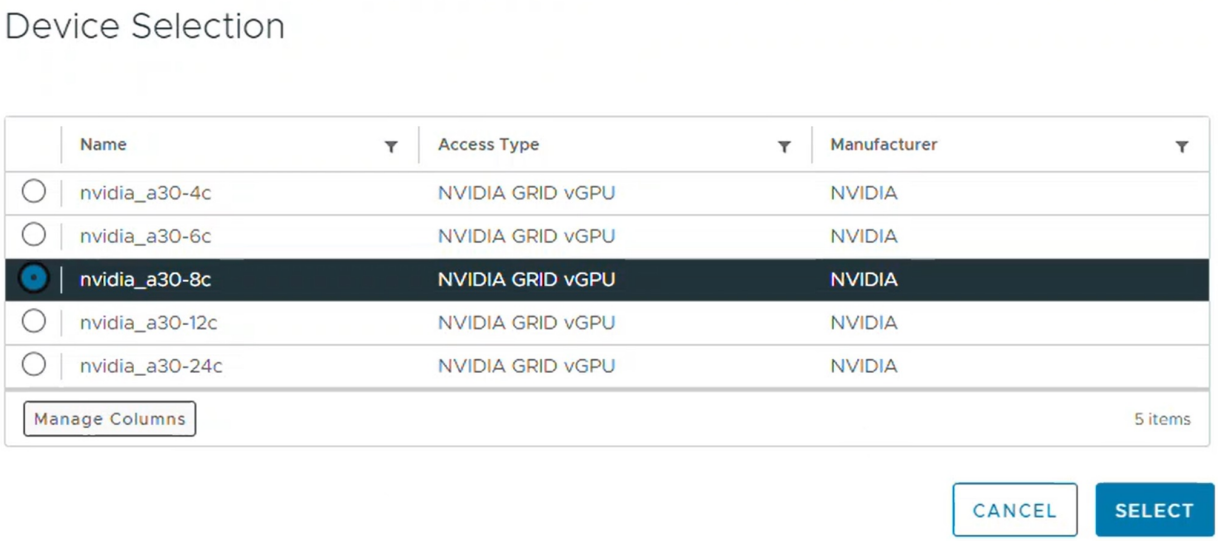
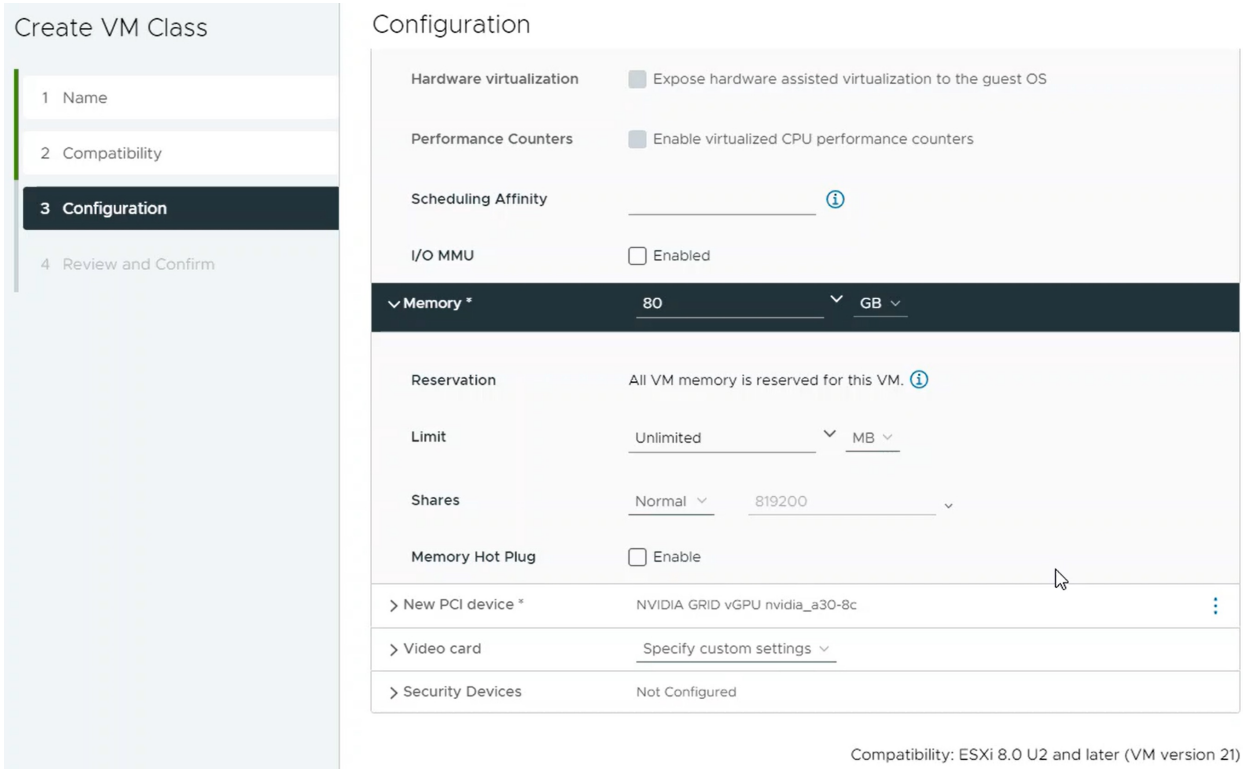


Abbildung 13-2. NVIDIA vGPU – Neues PCI-Gerät



Erstellen einer benutzerdefinierten VM-Klasse mit einem vGPU-Profil (v8 U2 und früher)

Im nächsten Schritt erstellen Sie eine benutzerdefinierte VM-Klasse mit einem vGPU-Profil. Das System verwendet diese Klassendefinition beim Erstellen der TKG-Clusterknoten.

Führen Sie die folgenden Anweisungen durch, um eine benutzerdefinierte VM-Klasse mit einem vGPU-Profil zu erstellen.

1 Melden Sie sich mit dem vSphere Client bei vCenter Server an.

2 Wählen Sie **Arbeitslastverwaltung** aus.

3 Wählen Sie **Dienste** aus.

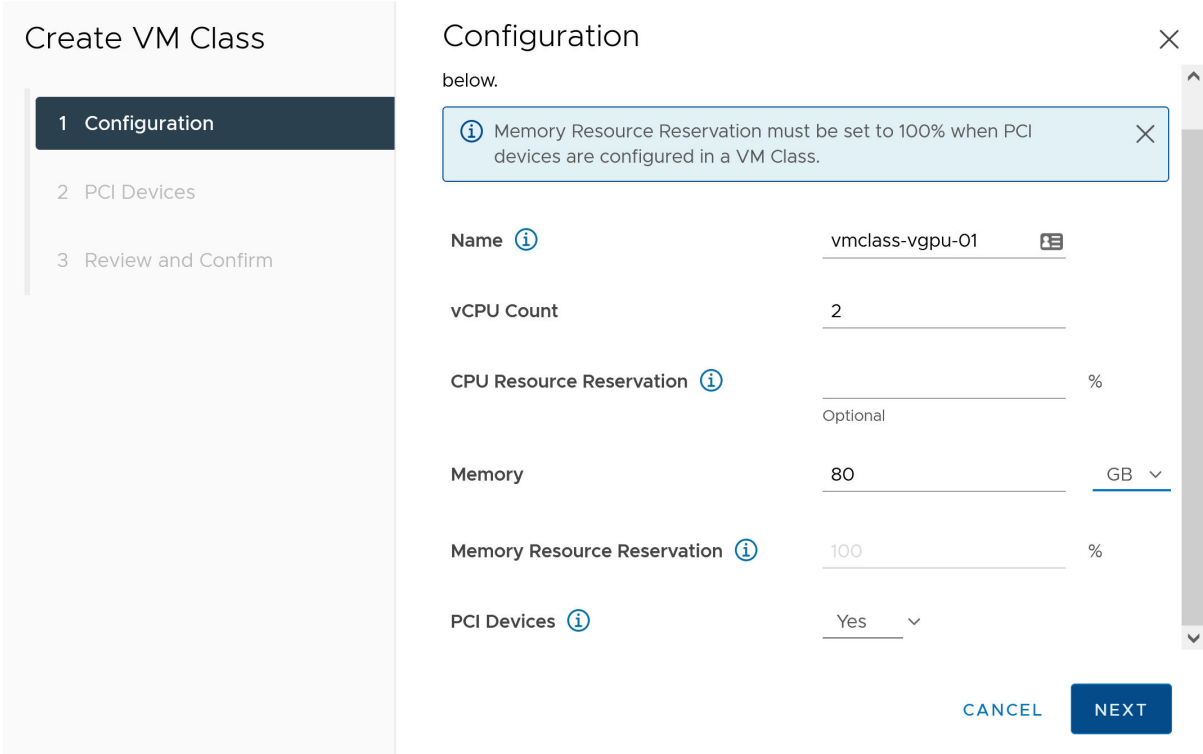
4 Wählen Sie **VM-Klassen** aus.

5 Klicken Sie auf **VM-Klasse erstellen**.

6 Konfigurieren Sie auf der Registerkarte **Konfiguration** die benutzerdefinierte VM-Klasse.

Konfigurationsfeld	Beschreibung
Name	Geben Sie einen selbsterklärenden Namen für die benutzerdefinierte VM-Klasse ein, wie z. B. **`vmclass-vgpu-1`**.
vCPU-Anzahl	**2**
CPU-Ressourcenreservierung	Optional, kann leer gelassen werden
Arbeitsspeicher	**80 GB**, z. B.
Arbeitsspeicher-Ressourcenreservierung	**100 %** (obligatorisch, wenn PCI-Geräte in einer VM-Klasse konfiguriert werden)
PCI-Geräte	**Ja** **Hinweis** Bei Auswahl von „Ja“ für PCI-Geräte wird das System darüber informiert, dass Sie ein GPU-Gerät verwenden. Darüber hinaus wird die VM-Klassenkonfiguration zur Unterstützung der vGPU-Konfiguration geändert. Weitere Informationen finden Sie unter Hinzufügen von PCI-Geräten zu einer VM-Klasse in vSphere with Tanzu.

Beispiel:



7 Klicken Sie auf **Weiter**.

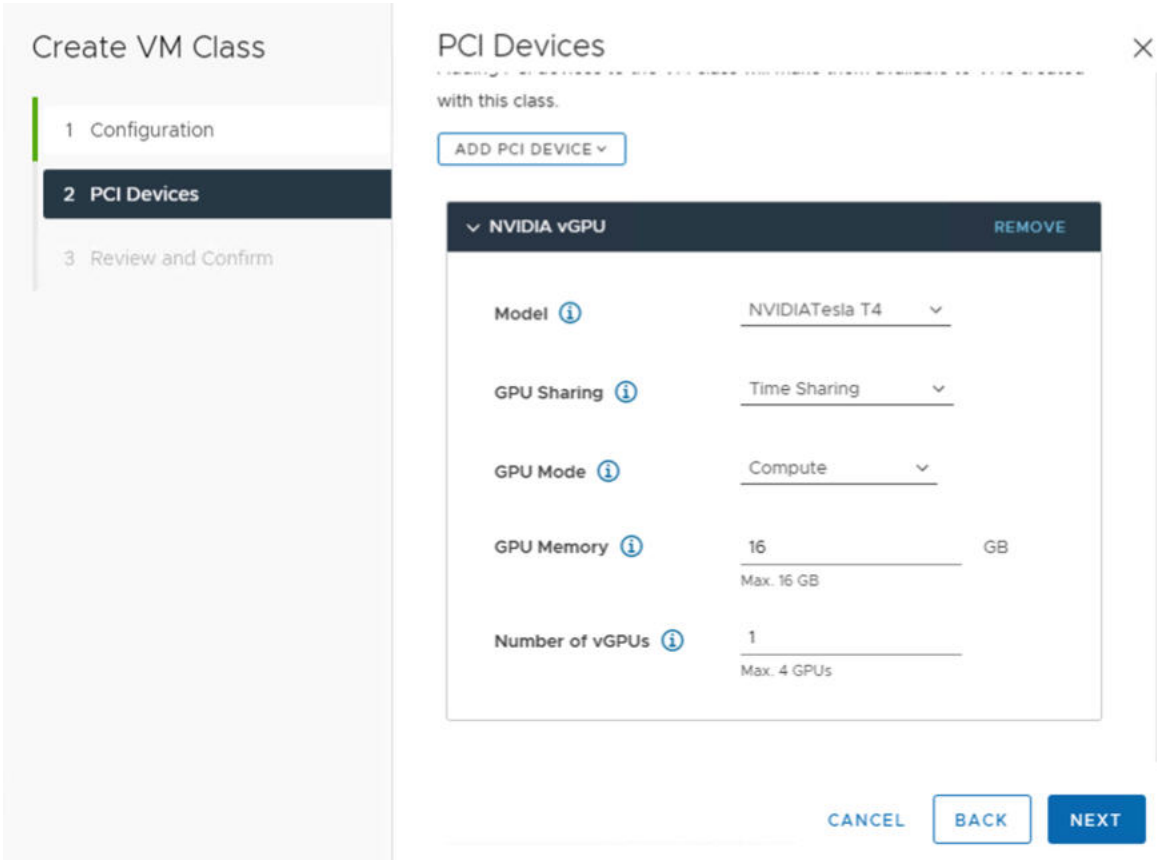
8 Wählen Sie auf der Registerkarte **PCI-Geräte** die Option **PCI-Gerät hinzufügen > NVIDIA vGPU** aus.

9 Konfigurieren Sie das NVIDIA vGPU-Modell.

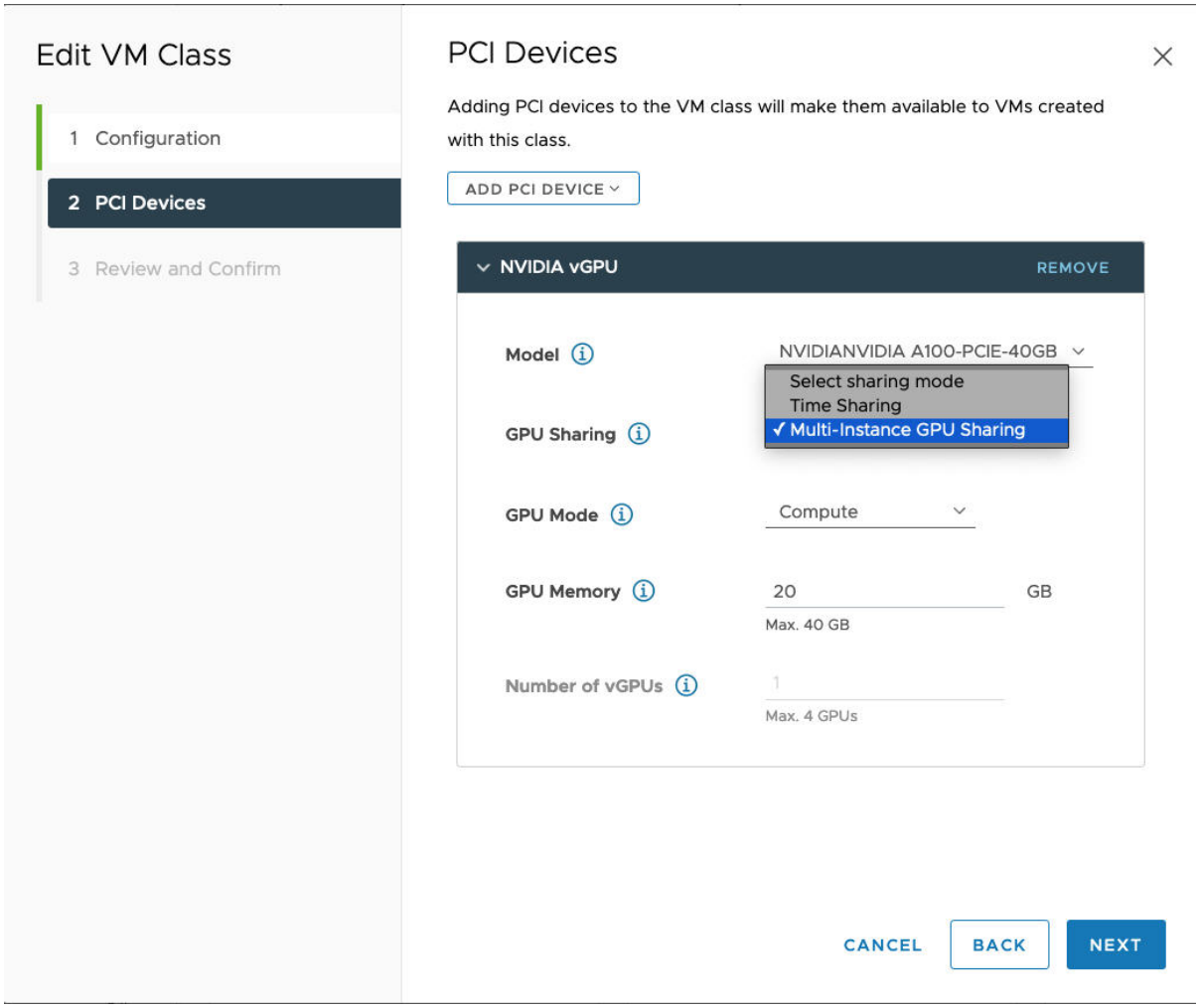
Feld „NVIDIA vGPU“	Beschreibung
Modell	Wählen Sie im Menü **NVIDIA vGPU > Modell** das Modell des NVIDIA GPU-Hardwaregeräts aus. Wenn im System keine Profile angezeigt werden, verfügt keiner der Hosts im Cluster über unterstützte PCI-Geräte.
GPU-Freigabe	Mit dieser Einstellung wird die Freigabe des GPU-Geräts für GPU-fähige VMs festgelegt. Zwei Arten von vGPU-Implementierungen stehen zur Verfügung: **Timesharing** und **GPU-Freigabe mehrerer Instanzen**. Im Modus „Timesharing“ weist der vGPU-Scheduler die GPU an, die Arbeit für jede vGPU-fähige VM fortlaufend über einen längeren Zeitraum durchzuführen, um die Leistung bestmöglich auf die vGPUs zu verteilen. Im MIG-Modus können mehrere vGPU-fähige VMs gleichzeitig auf einem einzelnen GPU-Gerät ausgeführt werden. Der MIG-Modus basiert auf einer neueren GPU-Architektur und wird nur auf NVIDIA A100- und A30-Geräten unterstützt. Wenn die MIG-Option nicht angezeigt wird, wird sie vom ausgewählten PCI-Gerät nicht unterstützt.
GPU-Modus	**Computing**

Feld „NVIDIA vGPU“	Beschreibung
GPU-Arbeitsspeicher	**8 GB**, z. B.
Anzahl der vGPUs	**1**, z. B.

Beispiel: Hierbei handelt es sich um ein NVIDIA vGPU-Profil, das im Modus „Timesharing“ konfiguriert ist:



Beispiel: Hierbei handelt es sich um ein NVIDIA vGPU-Profil, das im MIG-Modus mit einem unterstützten GPU-Gerät konfiguriert ist:



10 Klicken Sie auf **Weiter**.

11 Überprüfen und bestätigen Sie Ihre Auswahl.

12 Klicken Sie auf **Beenden**.

13 Stellen Sie sicher, dass die neue benutzerdefinierte VM-Klasse in der Liste der VM-Klassen verfügbar ist.

vGPU mit Dynamic DirectPath IO

Wenn Sie vGPU mit Dynamic DirectPath IO verwenden, führen Sie die folgende zusätzliche Konfiguration durch. Fügen Sie der benutzerdefinierten VM-Klasse, die Sie mit **Dynamic DirectPath IO** und dem ausgewählten unterstützten PCI-Gerät erstellt haben, eine zweite PCI-Gerätekonfiguration hinzu. Wenn eine VM-Klasse dieses Typs instanziiert wird, legt der vSphere Distributed Resource Scheduler (DRS) die VM-Platzierung fest.

1 Wählen Sie **Arbeitslastverwaltung** aus.

2 Wählen Sie **Dienste** aus.

3 Wählen Sie **VM-Klassen** aus.

4 Bearbeiten Sie die benutzerdefinierte VM-Klasse, die bereits mit dem **NVIDIA vGPU**-Profil konfiguriert ist.

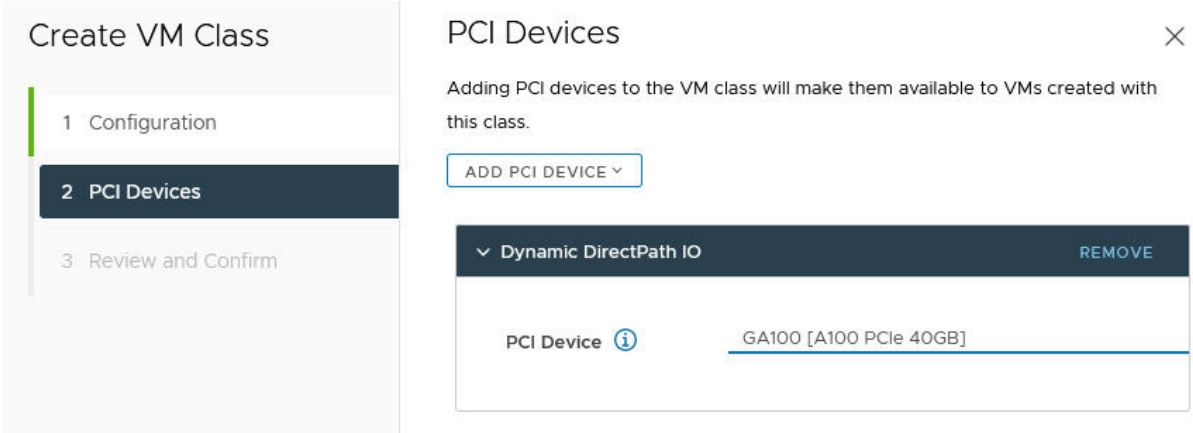
5 Wählen Sie die Registerkarte **PCI-Geräte** aus.

6 Klicken Sie auf **PCI-Gerät hinzufügen**.

7 Wählen Sie die Option **Dynamic DirectPath IO** aus.

8 Wählen Sie die Registerkarte **PCI-Gerät** aus.

Beispiel:



9 Klicken Sie auf **Weiter**.

10 Überprüfen und bestätigen Sie Ihre Auswahl.

11 Klicken Sie auf **Beenden**.

12 Stellen Sie sicher, dass die neue benutzerdefinierte VM-Klasse in der Liste der VM-Klassen verfügbar ist.

Verwenden privater Registrierungen mit TKG-Dienstclustern

14

Container-Registrierungen stellen Kubernetes-Operatoren ein praktisches Repository zum Speichern und Freigeben von Container-Images zur Verfügung. Dieser Abschnitt enthält eine Dokumentation zur Verwendung von privaten Registrierungen mit TKG-Dienstclustern.

Lesen Sie als Nächstes die folgenden Themen:

- Integrieren von TKG-Dienst-Clustern in eine private Containerregistrierung
- Erstellen eines geheimen Schlüssels für die private Registrierung
- Erstellen eines Pods aus einem Container-Image in einer privaten Registrierung
- Installieren des Supervisor-Diensts
- Konfigurieren eines Docker-Clients mit dem Harbor-Registrierungszertifikat
- Weitergeben von Standardpaketen an eine private Harbor-Registrierung

Integrieren von TKG-Dienst-Clustern in eine private Containerregistrierung

Informationen zum Integrieren von TKG-Dienst-Clustern in eine private Containerregistrierung finden Sie in diesem Thema.

Anwendungsbeispiel für eine private Container-Registrierung

Container-Registrierungen stellen eine kritische Funktion für Kubernetes-Bereitstellungen zur Verfügung und dienen als zentrales Repository für die Speicherung und Freigabe von Container-Images. Die am häufigsten verwendete Registrierung für öffentliche Container ist Docker Hub. Es gibt viele Private Container-Registrierungsangebote. VMware Harbor ist eine native Cloud-Registrierung mit Open Source und privatem Container, die im Lieferumfang von Supervisor enthalten ist.

Integration der privaten Container-Registrierung

Zur Integration einer privaten Registrierung in ein TKG-Dienst-Cluster konfigurieren Sie den Cluster mit einem oder mehreren selbstsignierten CA-Zertifikaten, um private Registrierungsinhalte über HTTPS abzuwickeln. Hierzu fügen Sie in der Clusterspezifikation ein Feld vom Typ `trust` mit dem Wert `additionalTrustedCAs` ein. Sie können eine beliebige Anzahl selbstsignierter CA-Zertifikate definieren, denen der TKGS-Cluster vertrauen soll. Mit dieser Funktion können Sie problemlos eine Liste von Zertifikaten definieren und diese Zertifikate bei Bedarf aktualisieren.

Sie können das Zertifikat der privaten Registrierung bei der erstmaligen Erstellung des Clusters konfigurieren. Alternativ können Sie auch einen vorhandenen Cluster aktualisieren und das Zertifikat der privaten Registrierung bereitstellen. Um einen vorhandenen Cluster zu bearbeiten und die Felder für das Zertifikat der privaten Registrierung hinzuzufügen, verwenden Sie die Methode `kubectl edit`. Weitere Informationen hierzu finden Sie unter Konfigurieren eines Texteditors für Kubectl.

Beachten Sie, dass sich die Implementierung des Felds `trust.additionalTrustedCAs` bei den unterstützten APIs für die Bereitstellung von TKGS-Clustern geringfügig unterscheidet. Weitere Informationen finden Sie in den Tabellen Tabelle 14-1. Vertrauenswürdige v1alpha3-API-Felder und Tabelle 14-2. Vertrauenswürdige v1beta1-API-Variable.

Beispiel für die v1alpha3-API

Das folgende Beispiel zeigt die Integration eines mit der v1alpha3-API bereitgestellten TKG-Dienst-Clusters in eine private Containerregistrierung mithilfe des zugehörigen CA-Zertifikats.

Bei Verwendung der TanzuKubernetesCluster v1alpha3-API weist das Feld `trust.additionalTrustedCAs` ein oder mehrere Name-Daten-Paare auf, von denen jedes ein TLS-Zertifikat für eine private Registrierung enthalten kann.

Tabelle 14-1. Vertrauenswürdige v1alpha3-API-Felder

Bereich	Beschreibung
`trust`	Abschnittsmarkierung. Akzeptiert keine Daten.
`additionalTrustedCAs`	Abschnittsmarkierung. Enthält ein Array von Zertifikaten mit den Feldern `name` und `data` für jedes Zertifikat.
`name`	Benutzerdefinierter Name des CA-Zertifikats.
`data`	Inhalt des CA-Zertifikats (`ca.crt`) im PEM-Format, das doppelt Base64-codiert ist. **Hinweis** Gemäß der v1alpha3-API muss der Zertifikatsinhalt einzeln Base64-codiert sein. Wenn der Inhalt nicht einzeln Base64-codiert ist, kann die resultierende PEM-Datei nicht verarbeitet werden.

Verwenden Sie das folgende Verfahren, um Harbor mithilfe des Harbor-Registrierungszertifikats in einen v1alpha3-API-Cluster zu integrieren.

1 Laden Sie das Harbor-**Registrierungszertifikat** von der Harbor-Webschnittstelle im Bildschirm **Projekte > Repositorys** herunter.

   Die Datei mit dem Zertifizierungsstellenzertifikat wird als `ca.crt` heruntergeladen.

2 Verwenden Sie für den Inhalt des CA-Zertifikats eine einzelne Base64-Codierung.

   - Linux: `base64 -w 0 ca.crt`
   - Windows: https://www.base64encode.org/.

3 Schließen Sie die Felder vom Typ `trust.additionalTrustedCAs` in die Clusterspezifikation ein und befüllen Sie sie mit den Werten `name` und `data`.

```
#tkc-with-trusted-private-reg-cert.yaml
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: tkc01
  namespace: tkgs-cluster-ns
spec:
   topology:
     controlPlane:
       replicas: 3
       storageClass: tkgs-storage-policy
       vmClass: guaranteed-medium
       tkr:
         reference:
           name: v1.25.7---vmware.3-fips.1-tkg.1
     nodePools:
     - name: nodepool-01
       replicas: 3
       storageClass: tkgs-storage-policy
       vmClass: guaranteed-medium
       tkr:
         reference:
           name: v1.25.7---vmware.3-fips.1-tkg.1
   settings:
     storage:
       defaultClass: tkgs-storage-policy
     network:
       cni:
         name: antrea
       services:
         cidrBlocks: ["198.51.100.0/12"]
       pods:
         cidrBlocks: ["192.0.2.0/16"]
       serviceDomain: cluster.local
       trust:
         additionalTrustedCAs:
```

```
        - name: CompanyInternalCA-1
          data: LS0tLS1C...LS0tCg==
        - name: CompanyInternalCA-2
          data: MTLtMT1C...MT0tPg==
```

4 Führen Sie zum Rotieren eines Zertifikats den Befehl `kubectl edit` für die Clusterspezifikation aus und aktualisieren Sie den Wert `trust.additionalTrustedCAs.data`. Initiieren Sie anschließend ein paralleles Update.

Beispiel für die v1beta1-API

Im folgenden Beispiel wird die Integration eines mit der v1beta1-API bereitgestellten TKG-Dienst-Clusters in eine private Containerregistrierung mithilfe des zugehörigen CA-Zertifikats beschrieben.

Zur Integration der Registrierung eines privaten Containers in einen mit der Cluster-API v1beta1 bereitgestellten TKGS-Cluster verwenden Sie die Variable trust und befüllen sie mit einem geheimen Kubernetes-Schlüssel, der das Zertifikat der privaten Registrierung enthält.

Tabelle 14-2. Vertrauenswürdige v1beta1-API-Variable

Bereich	Beschreibung
`trust`	Abschnittsmarkierung. Akzeptiert keine Daten.
`additionalTrustedCAs`	Abschnittsmarkierung. Enthält ein Array von Zertifikaten mit einem Namen für jedes Zertifikat.
`name`	Der benutzerdefinierte Name für das Zuordnungsfeld `data` im geheimen Kubernetes-Schlüssel, der das doppelt Base64-codierte CA-Zertifikat im PEM-Format enthält. **Hinweis** Gemäß der v1beta1-API muss der Zertifikatsinhalt doppelt Base64-codiert sein. Wenn der Inhalt nicht doppelt Base64-codiert ist, kann die resultierende PEM-Datei nicht verarbeitet werden.

Verwenden Sie das folgende Verfahren, um Harbor mithilfe des Harbor-Registrierungszertifikats in einen v1beta1-API-Cluster zu integrieren.

1 Laden Sie das Harbor-**Registrierungszertifikat** von der Harbor-Webschnittstelle im Bildschirm **Projekte** > **Repositorys** herunter.

   Die Datei mit dem Zertifizierungsstellenzertifikat wird als `ca.crt` heruntergeladen.

2 Verwenden Sie für den Inhalt des CA-Zertifikats eine doppelte Base64-Codierung.

   - Linux: `base64 -w 0 ca.crt | base64 -w 0`
   - Windows: https://www.base64encode.org/.

3 Erstellen Sie eine YAML-Datei für die geheime Kubernetes-Definition mit den folgenden Inhalten.

```
#additional-ca-1.yaml
apiVersion: v1
data:
  additional-ca-1: TFMwdExTMUNSGlSzZ3Jaa...VVNVWkpRMEMwdExTMHRDZz09
kind: Secret
metadata:
  name: cluster01-user-trusted-ca-secret
  namespace: tkgs-cluster-ns
type: Opaque
```

Dabei gilt:

- Bei dem Wert der `data`-Zuordnung des geheimen Schlüssels handelt es sich um eine benutzerdefinierte Zeichenfolge, die den Namen des CA-Zertifikats darstellt (in diesem Beispiel `additional-ca-1`), dessen Wert ein doppeltes Base64-codiertes Zertifikat ist.
- Versehen Sie den geheimen Schlüssel im Abschnitt `metadata` mit dem Namen *`CLUSTER-NAME`*`-user-trusted-ca-secret`, wobei *CLUSTER-NAME* als Name des Clusters fungiert. Dieser geheime Schlüssel muss im selben vSphere-Namespace wie der Cluster erstellt werden.

4 Erstellen Sie den geheimen Kubernetes-Schlüssel deklarativ.

```
kubectl apply -f additional-ca-1.yaml
```

5 Überprüfen Sie die Erstellung des geheimen Schlüssels.

```
kubeclt get secret -n tkgs-cluster-ns cluster01-user-trusted-ca-secret
NAME                                          TYPE     DATA   AGE
cluster01-user-trusted-ca-secret              Opaque   12     2d22h
```

6 Schließen Sie die Variable `trust` in die Clusterspezifikation ein, die auf den Namen der Datenzuordnung für den geheimen Schlüssel verweist, d. h. in diesem Beispiel auf `additional-ca-1`.

```
#cluster-with-trusted-private-reg-cert.yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: cluster01
  namespace: tkgs-cluster-ns
spec:
  clusterNetwork:
    services:
      cidrBlocks: ["198.52.100.0/12"]
    pods:
      cidrBlocks: ["192.101.2.0/16"]
    serviceDomain: "cluster.local"
  topology:
```

```
    class: tanzukubernetescluster
    version: v1.26.5+vmware.2-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      machineDeployments:
        - class: node-pool
          name: node-pool-01
          replicas: 3
    variables:
      - name: vmClass
        value: guaranteed-medium
      - name: storageClass
        value: tkgs-storage-profile
      - name: defaultStorageClass
        value: tkgs-storage-profile
      - name: trust
        value:
          additionalTrustedCAs:
          - name: additional-ca-1
```

7 Erstellen Sie zum Rotieren des Zertifikats einen neuen geheimen Schlüssel und bearbeiten Sie die Clusterspezifikation mit dem entsprechenden Wert. Auf diese Weise wird ein paralleles Update des Clusters ausgelöst.

Hinweis Änderungen am *CLUSTER-NAME*-user-trusted-ca-secret werden vom System nicht überwacht. Änderungen am Wert der data-Zuordnung spiegeln sich nicht im Cluster wider. Sie müssen einen neuen geheimen Schlüssel erstellen, dessen Angaben eine Zuordnung von name zu trust.additionalTrustCAs beschreiben.

Erstellen eines geheimen Schlüssels für die private Registrierung

Erstellen Sie einen geheimen Registrierungsschlüssel und eine Referenz in den Pod- und Bereitstellungsspezifikationen, damit Sie Container-Images ohne Fehler abrufen können.

Docker Hub benötigt ein Konto zum Abrufen von Images. Sie können einen geheimen Kubernetes-Schlüssel mit Ihren Docker Hub-Anmeldedaten erstellen und in Pods und Bereitstellungsspezifikationen auf diesen geheimen Schlüssel verweisen.

Dieser Ansatz kann auch für andere private Registrierungen verwendet werden.

Voraussetzungen

Installieren Sie Docker auf einem Ubuntu-Clientcomputer. Weitere Informationen hierzu finden Sie unter https://docs.docker.com/engine/install/ubuntu/.

Verfahren

1 Führen Sie docker version aus und stellen Sie sicher, dass Docker installiert ist.

2 Führen Sie `docker login -u USERNAME` aus.

3 Geben Sie Ihr Docker Hub-Kennwort ein.

4 Führen Sie `cat ~/.docker/config.json` aus.

5 Führen Sie den folgenden Befehl aus, um einen generischen geheimen Schlüssel mit dem Namen `regcred` zu erstellen, der Ihre Docker Hub-Anmeldedaten enthält.

```
kubectl create secret generic regcred \
    --from-file=.dockerconfigjson=/home/ubuntu/.docker/config.json \
    --type=kubernetes.io/dockerconfigjson
```

Sie sollten `secret/regcred created` sehen.

6 Überprüfen Sie den geheimen Schlüssel.

```
kubectl get secrets

NAME                  TYPE                                  DATA   AGE
default-token-w7wqk   kubernetes.io/service-account-token   3      6h28m
regcred               kubernetes.io/dockerconfigjson        1      3h22m
```

Verweisen Sie in der YAML auf den registrierten geheimen Schlüssel.

7 Verweisen Sie im Abschnitt `imagePullSecrets` des Pods oder der Bereitstellungsspezifikation für Container-Images auf den geheimen `regcred`-Schlüssel.

Beispiel:

```
apiVersion: v1
kind: Pod
metadata:
  name: ping-pod
  namespace: default
spec:
  containers:
  - image: busybox:1.34
    name: busybox
    command: ["ping", "-c"]
    args: ["1", "8.8.8.8"]
  imagePullSecrets:
  - name: regcred
  restartPolicy: Never
```

Erstellen eines Pods aus einem Container-Image in einer privaten Registrierung

Um Images aus einer privaten Containerregistrierung für einen TKG-Dienst-Cluster abzurufen, konfigurieren Sie die Arbeitslast-YAML mit den Details der privaten Registrierung.

Mit diesem Verfahren können Images aus einer privaten Containerregistrierung wie beispielsweise Harbor-Registrierung abgerufen werden. In diesem Beispiel erstellen wir eine Pod-Spezifikation, die ein in einer Harbor-Registrierung gespeichertes Image verwendet und den zuvor konfigurierten Geheimschlüssel zum Abrufen von Images nutzt.

Voraussetzungen

Erstellen Sie einen geheimen Schlüssel für die Registrierung. Siehe Erstellen eines geheimen Schlüssels für die private Registrierung.

Verfahren

1 Stellen Sie einen TKG-Cluster bereit.

Weitere Informationen hierzu finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl.

2 Melden Sie sich beim Cluster an.

Weitere Informationen hierzu finden Sie unter Herstellen einer Verbindung mit einem TKG-Dienst-Cluster als vCenter Single Sign-On-Benutzer mit Kubectl.

3 Erstellen Sie einen geheimen Schlüssel für die Registrierung.

Weitere Informationen hierzu finden Sie unter Erstellen eines geheimen Schlüssels für die private Registrierung.

4 Erstellen Sie eine Beispiel-Pod-Spezifikation mit den Details zur privaten Registrierung.

`pod-example.yaml`

```
apiVersion: v1
kind: Pod
metadata:
  name: <workload-name>
  namespace: <kubernetes-namespace>
spec:
  containers:
  - name: private-reg-container
    image: <Registry-IP-or-FQDN>/<vsphere-namespace>/<image-name>:<version>
  imagePullSecrets:
  - name: <registry-secret-name>
```

- Ersetzen Sie `<workload-name>` durch den Namen der Pod-Arbeitslast.
- Ersetzen Sie `<kubernetes-namespace>` durch den Kubernetes-Namespace im Cluster, in dem der Pod erstellt werden soll. Hierbei muss es sich um denselben Kubernetes-Namespace handeln, in dem der Geheimschlüssel zum Ziehen von Images für den Registrierungsdienst im Tanzu Kubernetes-Cluster (z. B. dem Standard-Namespace) gespeichert ist.
- Ersetzen Sie `<Registry-IP-or-FQDN>` durch die IP-Adresse oder den FQDN für die eingebettete Harbor-Registrierung-Instanz, die auf dem Supervisor ausgeführt wird.

- Ersetzen Sie *<vsphere-namespace>* durch den vSphere-Namespace, in dem der Ziel-TKG-Cluster bereitgestellt wird.
- Ersetzen Sie `<image-name>` durch einen Image-Namen Ihrer Wahl.
- Ersetzen Sie `<version>` durch eine geeignete Version des Images, z. B. „neueste“.
- Ersetzen Sie `<registry-secret-name>` durch den Namen des Ihnen zuvor erstellten Geheimschlüssel zum Ziehen von Images .

5 Erstellen Sie die Arbeitslast mithilfe der von Ihnen definierten Pod-Spezifikation.

```
kubectl apply -f pod-example.yaml
```

Der Pod sollte aus dem Image erstellt werden, das aus der Registrierung gezogen wurde.

Installieren des Supervisor-Diensts

Sie können die Harbor-Container-Registrierung als Supervisor-Dienst installieren und Harbor als private Registrierung ausführen.

Voraussetzungen

Beachten Sie die folgenden Voraussetzungen:

- vSphere 8 Supervisor ist aktiviert.
- Sie haben sich mit Supervisor-Diensten vertraut gemacht.

Hinweis Diese Anweisungen werden mit vSphere 8- und NSX 4-Netzwerk validiert.

Herunterladen der erforderlichen YAML-Dateien

Laden Sie die erforderlichen YAML-Dateien herunter, einschließlich Contour und Harbor.

1 Laden Sie die folgenden Contour-Dateien von der Site für den Kubernetes-Ingress-Controller-Dienst herunter:

   a Die Dienstdefinitionsdatei für Contour: `contour.yml`

   b Die Dienstkonfigurationsdatei für Contour: `contour-data-values.yml`

2 Laden Sie die folgenden Harbor-Dateien von der Site für den cloudnativen Registrierungsdienst herunter:

   a Die Dienstdefinitionsdatei für Harbor: `harbor.yml`

   b Die Dienstkonfigurationsdatei für Harbor: `harbor-data-values.yml`

Installieren von Contour

Vor der Installation von Harbor müssen Sie zuerst Contour installieren.

1 Laden Sie `contour.yml` unter **Arbeitslastverwaltung > > Dienste > Hinzufügen** in vCenter hoch.

2 Vergewissern Sie sich, dass die Dienstdefinition für Contour hinzugefügt wurde.

3 Wählen Sie **Arbeitslastverwaltung > Supervisoren > Supervisor > Konfigurieren** aus.

4 Wählen Sie **Supervisor-Dienste > Überblick** aus.

5 Wählen Sie die Registerkarte **Verfügbar** aus.

6 Wählen Sie **Contour** aus und klicken Sie auf **Installieren**.

7 Kopieren Sie den Inhalt aus `contour-data-values.yml` und fügen Sie ihn in das Eingabefeld „YAML-Dienstkonfiguration“ ein.

Hinweis Die Contour-Datenwerte können unverändert verwendet werden. Änderungen an der Konfiguration sind nicht erforderlich. Der Diensttyp für Envoy ist auf LoadBalancer festgelegt.

8 Klicken Sie auf **OK**, um mit der Installation von Contour fortzufahren.

9 Stellen Sie sicher, dass Contour installiert ist.

a Wählen Sie den vSphere-Namespace mit dem Namen *svc-contour-domain-XXXX* aus.

b Klicken Sie auf die Registerkarte **Netzwerk** und dann auf **Dienste**.

c Der Contour-Dienst vom Typ „ClusterIP“ und der Envoy-Dienst vom Typ „LoadBalancer“ sollten angezeigt werden. Der envoy-Dienst sollte über eine externe IP-Adresse verfügen.

Aktualisieren von Harbor-Datenwerten

Aktualisieren Sie vor der Installation von Harbor die Datei mit den Datenwerten.

1 Öffnen Sie mit einem Texteditor die Datei `harbor-data-values.yml`.

2 Nehmen Sie mindestens die folgenden Änderungen vor (die Bearbeitung anderer Felder ist optional).

3 Speichern Sie die Änderungen.

Name	Wert
hostname	harbordomain.com (Wählen Sie einen eindeutigen Wert aus.)
tlsCertificate.tlsSecretLabels	{"managed-by": "vmware-vRegistry"} (Überprüfen Sie diesen Wert, behalten Sie ihn jedoch bei.)
persistence.persistentVolumeClaim.registry.storageClass	"vwt-storage-policy" (geben Sie den Namen der vSphere Speicherrichtlinie für Supervisor ein)

Name	Wert
persistence.persistentVolumeClaim.jobservice.storageClass	"vwt-storage-policy" (geben Sie den Namen der vSphere Speicherrichtlinie für Supervisor ein)
persistence.persistentVolumeClaim.database.storageClass	"vwt-storage-policy" (geben Sie den Namen der vSphere Speicherrichtlinie für Supervisor ein)
persistence.persistentVolumeClaim.redis.storageClass	"vwt-storage-policy" (geben Sie den Namen der vSphere Speicherrichtlinie für Supervisor ein)
persistence.persistentVolumeClaim.trivy.storageClass	"vwt-storage-policy" (geben Sie den Namen der vSphere Speicherrichtlinie für Supervisor ein)

Installieren von Harbor

Installieren Sie Harbor, indem Sie die folgenden Anweisungen ausführen.

1 Laden Sie `harbor.yml` unter **Arbeitslastverwaltung > > Dienste > Hinzufügen** in vCenter hoch.

2 Vergewissern Sie sich, dass die Dienstdefinition für Harbor hinzugefügt wurde.

3 Wählen Sie **Arbeitslastverwaltung > Supervisoren > Supervisor > Konfigurieren** aus.

4 Wählen Sie **Supervisor-Dienste > Überblick** aus.

5 Wählen Sie die Registerkarte **Verfügbar** aus.

6 Wählen Sie **Harbor** aus und klicken Sie auf **Installieren**.

7 Kopieren Sie den bearbeiteten Inhalt aus `harbor-data-values.yml` und fügen Sie ihn in das Eingabefeld „YAML-Dienstkonfiguration“ ein.

8 Klicken Sie auf **OK**, um mit der Installation von Harbor fortzufahren.

9 Stellen Sie sicher, dass Harbor installiert ist.

   a Wählen Sie den vSphere-Namespace mit dem Namen *svc-harbor-domain-XXXX* aus.

   b Klicken Sie auf die Registerkarte **Netzwerk** und dann auf **Dienste**.

   c Es sollten mehrere installierte Container für Harbor angezeigt werden (jeder ein Dienst vom Typ „ClusterIP“).

Konfigurieren von DNS für Harbor

Sie müssen einen Domänennamen registrieren und einen DNS-Datensatz für Harbor einrichten.

1 Wählen Sie **Arbeitslastverwaltung > Namespaces** aus.

2 Wählen Sie den Namespace **Contour** aus.

3 Wählen Sie **Netzwerk > Dienste** aus.

4 Notieren Sie sich die externe IP-Adresse für den Envoy-Ingress-Dienst, z. B. 10.197.154.71.

5 Registrieren Sie den Harbor-Domänennamen (FQDN), den Sie in der Harbor-Konfiguration angegeben haben.

6 Erstellen Sie mit AWS Route 53 oder einem ähnlichen Dienst einen DNS-Datensatz „A“.

Anmelden bei Harbor

Nach der Einrichtung des Harbor-DNS melden Sie sich an.

1 Wechseln Sie zu dem Domänennamen, den Sie für Harbor registriert haben.

2 Melden Sie sich mit „admin“ und dem Kennwort, das Sie in der Harbor-Konfiguration angegeben haben, bei der Domäne an.

3 Ändern Sie das Kennwort nach der Anmeldung in ein sichereres Kennwort.

Konfigurieren des Supervisors, sodass er der Harbor-Registrierung vertraut (optional)

TKG-Cluster werden automatisch so konfiguriert, dass sie dem Harbor-Supervisor-Dienst vertrauen, wenn sowohl der TKG-Cluster als auch Harbor auf demselben Supervisor bereitgestellt werden. Allerdings wird Supervisor beim Erstellen von vSphere-Pods NICHT automatisch so konfiguriert, dass dem Harbor-Supervisor-Dienst vertraut wird. Führen Sie die folgenden Schritte aus, um eine Vertrauensstellung zwischen Supervisor und dem Harbor-Dienst herzustellen, indem Sie die ConfigMap mit dem Harbor-CA-Zertifikat aktualisieren.

1 Navigieren Sie in Harbor zu **Verwaltung > Konfiguration > Systemeinstellungen**.

2 Laden Sie das Root-Zertifikat für die Registrierung herunter. Dabei handelt es sich um eine Datei mit dem Namen `ca.crt`.

3 Konfigurieren Sie die KUBE_EDITOR-Umgebungsvariable.

   Weitere Informationen hierzu finden Sie unter Konfigurieren eines Texteditors für Kubectl.

4 Melden Sie sich mithilfe von kubectl bei Supervisor an.

   Weitere Informationen hierzu finden Sie unter Herstellen einer Verbindung zu Supervisor als vCenter Single Sign-On-Benutzer mit Kubectl.

5 Führen Sie einen Kontextwechsel zum Supervisor-Kontext (IP-Adresse) durch.

6 Bearbeiten Sie configmap/image-fetch-ca-bundle mit folgendem Befehl:

```
kubectl edit configmap image-fetcher-ca-bundle -n kube-system
```

7 Kopieren Sie den Inhalt der Harbor-Datei „ca.crt“ und hängen Sie die ConfigMap unter dem vorhandenen Zertifikat für Supervisor an (dieses darf nicht geändert werden). Speichern Sie die an der Datei vorgenommenen Änderungen. Kubectl sollte jetzt „configmap/image-fetcher-ca-bundle edited“ melden.

Konfigurieren eines Docker-Clients mit dem Harbor-Registrierungszertifikat

Um mit Container-Images in einer Registrierung mithilfe von Docker zu arbeiten, müssen Sie Ihrem Docker-Client das Registrierungszertifikat hinzufügen. Das Zertifikat wird zur Authentifizierung von Docker bei der Anmeldung in der Registrierung verwendet.

Konfigurieren Sie Ihren Docker-Client für die Interaktion mit einer Containerregistrierung, z. B. Harbor-Registrierung oder Docker Hub. In diesem Beispiel wird davon ausgegangen, dass Sie den Harbor-Supervisor-Dienst verwenden.

Voraussetzungen

Bei dieser Aufgabe wird davon ausgegangen, dass Sie einen Linux-Host (Ubuntu) verwenden, auf dem der Docker-Daemon installiert ist. Informationen zum Installieren der Docker-Engine (Daemon) auf einem Ubuntu-Host finden Sie unter https://docs.docker.com/engine/install/ubuntu/.

Führen Sie den folgenden Befehl aus, um zu überprüfen, ob Docker installiert ist und Sie Images aus dem Docker-Hub ziehen können:

```
docker run hello-world
```

Erwartetes Ergebnis:

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Hinweis Diese Anweisungen werden mithilfe von Ubuntu 20.04 und Docker 19.03 verifiziert.

Verfahren

1. Melden Sie sich bei der Harbor-Registrierung an.
2. Wählen Sie **Verwaltung > Konfiguration > Root-Zertifikat für die Registrierung** aus.
3. Klicken Sie auf **Herunterladen**, um das Harbor-Registrierungszertifikat mit dem Namen `ca.crt` herunterzuladen.

   Hinweis Sofern nötig, ändern Sie den Namen des Zertifikats zu `ca.crt`.
4. Kopieren Sie die Datei `ca.crt` sicher auf Ihren Docker-Hostclient.
5. Erstellen Sie auf dem Docker-Host einen Verzeichnispfad für die Privatregistrierung mithilfe der Harbor-IP-Adresse.

   ```
   /etc/docker/certs.d/IP-address-or-FQDN-of-harbor/
   ```

   Beispiel:

   ```
   mkdir /etc/docker/certs.d/10.179.145.77
   ```

6 Verschieben Sie die Datei `ca.crt` in dieses Verzeichnis.

Beispiel:

```
mv ca.crt /etc/docker/certs.d/10.179.145.77/ca.crt
```

7 Starten Sie den Docker-Daemon neu.

```
sudo systemctl restart docker.service
```

8 Melden Sie sich beim eingebetteten Harbor-Registrierung mit Ihrem Docker-Client an.

```
docker login https://10.179.145.77
```

Sie sollten folgende Meldung sehen:

```
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Weitergeben von Standardpaketen an eine private Harbor-Registrierung

In diesem Thema finden Sie Informationen zum Weitergeben der Standardpakete aus der Registrierung für öffentliche Container in eine private Harbor-Registrierung, die als Supervisor-Dienst ausgeführt wird.

Voraussetzungen

Erfüllen Sie die folgenden Voraussetzungen

- Installieren des Supervisor-Diensts.
- Konfigurieren Sie einen Ubuntu-Host als Docker-Client. Weitere Informationen hierzu finden Sie unter Konfigurieren eines Docker-Clients mit dem Harbor-Registrierungszertifikat.
- Installieren Sie jq auf dem Ubuntu-Host. Weitere Informationen hierzu finden Sie unter https://jqlang.github.io/jq/download/.

Installieren des Carvel-Dienstprogramms „Image-Paket“

Installieren Sie das Carvel-Dienstprogramm `imgpkg` auf dem Ubuntu-Client, auf dem kubectl installiert ist.

1 Installieren Sie `imgpkg`.

```
wget -O- https://carvel.dev/install.sh > install.sh
sudo bash install.sh
```

2 Überprüfen Sie die Installation.

```
imgpkg version
```

Erwartetes Ergebnis:

```
imgpkg version 0.41.1
```

Auflisten der verfügbaren Images für jedes Tanzu-Standardpaket

Cert Manager

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/cert-manager
```

Contour mit Envoy

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/contour
```

ExternalDNS

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/external-dns
```

Prometheus mit Alertmanager

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/prometheus
```

Grafana

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/grafana
```

Fluent Bit

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/fluent-bit
```

Harbor

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/harbor
```

Abrufen der Standardpaket-Images aus der öffentlichen VMware-Registrierung

Rufen Sie die Tanzu-Standardpakete aus der öffentlichen VMware-Registrierung unter https://projects.registry.vmware.com/ ab. Aktualisieren Sie die Version, sodass sie mit der abzurufenden Version übereinstimmt.

Cert Manager

```
docker pull projects.registry.vmware.com/tkg/packages/standard/cert-manager:v1.7.2_vmware.3-tkg.3
```

Contour mit Envoy

```
docker pull projects.registry.vmware.com/tkg/packages/standard/contour:v1.23.5_vmware.1-tkg.1
```

ExternalDNS

```
docker pull projects.registry.vmware.com/tkg/packages/standard/external-dns:v0.12.2_vmware.5-tkg.1
```

Prometheus mit Alertmanager

```
docker pull projects.registry.vmware.com/tkg/packages/standard/prometheus:v2.37.0_vmware.3-tkg.1
```

Grafana

```
docker pull projects.registry.vmware.com/tkg/packages/standard/grafana:v7.5.17_vmware.2-tkg.1
```

Fluent Bit

```
docker pull projects.registry.vmware.com/tkg/packages/standard/fluent-bit:v1.9.5_vmware.1-tkg.2
```

Harbor

```
docker pull projects.registry.vmware.com/tkg/packages/standard/harbor:v2.7.1_vmware.1-tkg.1
```

Erstellen eines Projekts in der privaten Harbor-Registrierung

Erstellen Sie ein öffentliches Projekt in Harbor, um die Tanzu-Pakete zu hosten.

1 Melden Sie sich bei der privaten Harbor-Registrierung an.

2 Wählen Sie in Harbor die Option **Projekte > Neues Projekt** aus.

3 Erstellen Sie ein neues öffentliches Projekt mit dem Namen **tanzu-packages**.

Kennzeichnen der Standardpaket-Images

Kennzeichnen Sie die Images mit der folgenden Syntax.

```
docker tag SOURCE_IMAGE[:TAG] harbordomain.com/tanzu-packages/REPOSITORY[:TAG]
```

Dabei gilt:

- SOURCE_IMAGE[:TAG] ist der Name des Image, das Sie abgerufen haben.
- *harbordomain.com* ist der DNS-Name Ihres Harbor-Servers.
- REPOSITORY[:TAG] ist der Name des Image-Tags

Cert Manager

```
docker tag projects.registry.vmware.com/tkg/packages/standard/cert-manager:v1.7.2_vmware.3-tkg.3 harbordomain.com/tanzu-packages/cert-manager:v1.7.2
```

Contour mit Envoy

```
docker tag projects.registry.vmware.com/tkg/packages/standard/contour:v1.23.5_vmware.1-tkg.1 harbordomain.com/tanzu-packages/contour:v1.23.5
```

ExternalDNS

```
docker tag projects.registry.vmware.com/tkg/packages/standard/external-dns:v0.12.2_vmware.5-tkg.1 harbordomain.com/tanzu-packages/external-dns:v0.12.2
```

Prometheus mit Alertmanager

```
docker tag projects.registry.vmware.com/tkg/packages/standard/prometheus:v2.37.0_vmware.3-tkg.1 harbordomain.com/tanzu-packages/prometheus:v2.37.0
```

Grafana

```
docker tag projects.registry.vmware.com/tkg/packages/standard/grafana:v7.5.17_vmware.2-tkg.1 harbordomain.com/tanzu-packages/grafana:v7.5.17
```

Fluent Bit

```
docker tag projects.registry.vmware.com/tkg/packages/standard/fluent-bit:v1.9.5_vmware.1-tkg.2 harbordomain.com/tanzu-packages/fluent-bit:v1.9.5
```

Harbor

```
docker tag projects.registry.vmware.com/tkg/packages/standard/harbor:v2.7.1_vmware.1-tkg.1 harbordomain.com/tanzu-packages/harbor:v2.7.1
```

Weitergeben der Standardpaket-Images an die private Harbor-Registrierung

Geben Sie die Images mit der folgenden Syntax weiter.

```
docker push harbordomain.com/tanzu-packages/PACKAGE
```

Dabei gilt:

- *harbordomain.com* ist der DNS-Name Ihres Harbor-Servers.
- *tanzu-packages* ist der Name des Harbor-Projekts.
- *PACKAGE* ist der Name des Tanzu-Pakets.
- *vX.X.X* ist die Tag-Version des Pakets.

Cert Manager

```
docker push harbordomain.com/tanzu-packages/cert-manager:v1.7.2
```

Contour mit Envoy

```
docker push harbordomain.com/tanzu-packages/contour:v1.23.5
```

ExternalDNS

```
docker push harbordomain.com/tanzu-packages/external-dns:v0.12.2
```

Prometheus mit Alertmanager

```
docker push harbordomain.com/tanzu-packages/prometheus:v2.37.0
```

Grafana

```
docker push harbordomain.com/tanzu-packages/grafana:v7.5.17
```

Fluent Bit

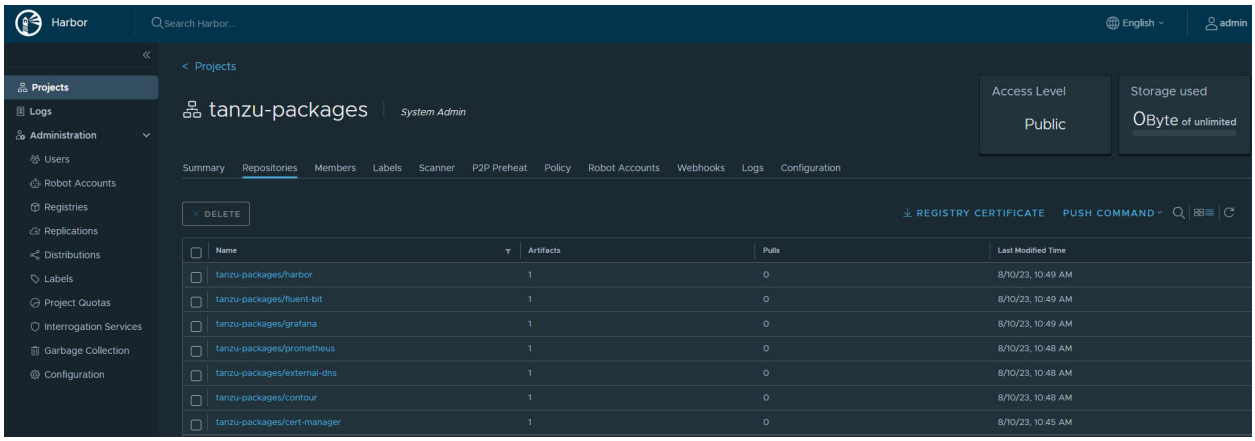
```
docker push harbordomain.com/tanzu-packages/fluent-bit:v1.9.5
```

Harbor

```
docker push harbordomain.com/tanzu-packages/harbor:v2.7.1
```

Nachdem Sie die Images an die private Harbor-Registrierung weitergegeben haben, stellen Sie sicher, dass jedes Image in der Harbor-Webschnittstelle angezeigt wird.

Abbildung 14-1. Tanzu-Standardpakete in der privaten Harbor-Registrierung



Abrufen eines Container-Images für die Bereitstellung

Um zu überprüfen, ob sich die Images in der privaten Harbor-Registrierung befinden, rufen Sie ein Image mit der folgenden Syntax von dort ab:

```
docker pull harbordomain.com/tanzu-packages/PACKAGE:TAG
```

Beispiel:

```
docker pull harbordomain.com/tanzu-packages/fluent-bit:v1.9.5
```

Sie können cURL verwenden, um die Harbor-API aufzurufen und die Pakete aufzulisten. Laden Sie dazu das Harbor-Zertifikat über die Harbor-Webschnittstelle herunter und führen Sie dann den folgenden Befehl aus:

```
curl -X 'GET' 'https://harbordomain.com/api/v2.0/projects/tanzu-packages/repositories?
page=1&page_size=-1' -H 'accept: application/json' --cacert ca.crt | jq '.[].name'
```

Der Befehl gibt die verfügbaren tanzu-packages zurück.

```
"tanzu-packages/harbor"
"tanzu-packages/fluent-bit"
"tanzu-packages/grafana"
"tanzu-packages/prometheus"
"tanzu-packages/external-dns"
"tanzu-packages/contour"
"tanzu-packages/cert-manager"
```

Erstellen von Snapshots in einem TKG-Dienst-Cluster

15

TKG-Dienst-Cluster unterstützen Volume-Snapshot- und Wiederherstellungsfunktionen. Als DevOps-Benutzer können Sie Arbeitslasten in TKG-Clustern schützen, indem Sie Volume-Snapshots erstellen.

Mithilfe eines Snapshots können Sie ein neues Volume bereitstellen, das bereits mit den Snapshot-Daten aufgefüllt ist.

Voraussetzungen

Zum Erstellen von CSI-Snapshots für TKG-Dienst-Cluster muss Ihre Umgebung folgende Voraussetzungen erfüllen.

- vSphere 8.0 Update 2 oder höher.
- Tanzu Kubernetes-Version, das CSI-Snapshots mit TKr v1.26.5 oder höher für vSphere 8.0.2 oder höher unterstützt. Weitere Informationen hierzu finden Sie in den Versionshinweisen zu den VMware Tanzu Kubernetes-Versionen.
- Aktuelle kompatible Supervisor-Version. Weitere Informationen hierzu finden Sie unter Versionshinweise zu VMware vSphere with Tanzu 8.0.

Anforderungen

Die CSI-Snapshot-Funktion wird als TKG-Paket bereitgestellt. Folgende Anforderungen gelten für die Verwendung des CSI-Snapshot-Pakets:

- Verwenden Sie die Repository-Version v2023.9.19 oder höher des TKG-Standardpakets. Weitere Informationen finden Sie unter Repository-Versionen des Tanzu-Standardpakets.
- Installieren Sie das Cert Manager-Paket. Weitere Informationen finden Sie unter Installieren und Verwenden von VMware Tanzu-Paketen.
- Verwenden Sie die Tanzu-CLI, um vsphere-pv-csi-webhook zu installieren und bereitzustellen. Weitere Informationen hierzu finden Sie unter Installieren und Bereitstellen des vSphere PVCSI Webhooks.
- Verwenden Sie die Tanzu-CLI, um external-csi-snapshot-webhook zu installieren und bereitzustellen. Weitere Informationen hierzu finden Sie unter Installieren und Bereitstellen des externen Webhooks für CSI-Snapshots.

Richtlinien und Einschränkungen

Wenn Sie die Snapshot- und Wiederherstellungsfunktionen mit TKG-Clustern verwenden, befolgen Sie die folgenden Richtlinien.

- Nur Block-Volumes unterstützen Volume-Snapshot- und Wiederherstellungsvorgänge. Sie können diese Vorgänge nicht bei vSphere-Datei-Volumes verwenden.
- Wenn Sie eine PVC aus einem VolumeSnapshot erstellen, sollte sich dieser im selben Datenspeicher wie der ursprüngliche VolumeSnapshot befinden. Andernfalls schlägt die Bereitstellung dieser PVC mit folgendem Fehler fehl:

  ```
  failed to provision volume with StorageClass <storage-class-name>: rpc error: code =
  Internal desc = failed to create volume. Error: failed to get the compatible datastore
  for create volume from snapshot <snapshot-name> with error: failed to find datastore with
  URL <datastore-url> from the input datastore list, <[datastore-list]>
  ```

  Der Datenspeicher für die Ziel-PVC, die Sie aus dem VolumeSnapshot erstellen, wird durch die StorageClass in der PVC-Definition bestimmt. Stellen Sie sicher, dass die StorageClass der Ziel-PVC und die StorageClass der ursprünglichen Quell-PVC auf denselben Datenspeicher – den Datenspeicher der Quell-PVC – verweisen. Diese Regel gilt auch für die Topologieanforderungen in den StorageClass-Definitionen. Die Anforderungen müssen ebenfalls auf denselben gemeinsamen Datenspeicher verweisen. Alle in Konflikt stehenden Topologieanforderungen führen zu demselben Fehler (siehe oben).
- Sie können ein Volume, das zugeordnete Snapshots enthält, nicht löschen oder erweitern. Löschen Sie alle Snapshots, um das Quell-Volume zu erweitern oder zu löschen.
- Wenn Sie ein Volume aus einem Snapshot erstellen, stellen Sie sicher, dass die Größe des Volumes auf die Größe des Snapshots abgestimmt ist.
- Die Überwachung des Speicherkontingents wird für Snapshots nicht unterstützt.
- Sie können in der vSphere-Konfiguration nicht die maximale Anzahl von Snapshots pro Volume konfigurieren. Verwenden Sie zwei bis drei Snapshots pro virtueller Festplatte, um eine bessere Leistung zu erzielen. Weitere Informationen finden Sie unter Best practices for using VMware snapshots in the vSphere environment (Best Practices für die Verwendung von VMware-Snapshots in der vSphere-Umgebung).

  Verwenden Sie für vSAN ESA maximal 32 Snapshots pro Volume. Weitere Informationen zu vSAN ESA finden Sie unter vSAN Express Storage Architecture.

Lesen Sie als Nächstes die folgenden Themen:

- Installieren und Bereitstellen des externen Webhooks für CSI-Snapshots
- Installieren und Bereitstellen des vSphere PVCSI Webhooks
- Erstellen von Snapshots in einem TKG-Dienst-Cluster

Installieren und Bereitstellen des externen Webhooks für CSI-Snapshots

Zum Verwenden der Snapshot-Technologie in TKG-Dienst-Clustern müssen Sie einen externen Webhook für CSI-Snapshots in einem TKG-Cluster installieren und bereitstellen. Der externe Webhook für CSI-Snapshots ist eine Open Source-Komponente mit einem HTTP-Callback, das auf Zugangsanforderungen reagiert. Er ist für die Validierung von Volume-Snapshot-Objekten verantwortlich.

Der externe Webhook für CSI-Snapshots wird automatisch in Supervisor installiert. Dieses Thema gilt nur für TKG-Dienst-Cluster.

Voraussetzungen

- Sie verfügen über einen ausgeführten Supervisor.
- Tanzu-CLI und kubectl sind installiert. Weitere Informationen finden Sie unter Installieren von CLI-Tools für TKG-Dienst-Cluster.
- Sie haben sich beim TKG-Cluster angemeldet. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu TKG-Dienst-Clustern mithilfe der vCenter SSO-Authentifizierung.

Vorbereiten eines TKG-Dienst-Clusters für die Installation des externen Webhooks für CSI-Snapshots

Führen Sie die folgenden Schritte aus, um den externen Webhook für CSI-Snapshots in einem TKG-Dienst-Cluster bereitzustellen.

Verfahren

1. Rufen Sie die Admin-Anmeldedaten des TKG-Clusters ab, in dem Sie den externen Webhook für CSI-Snapshots bereitstellen möchten.

   ```
   tanzu cluster kubeconfig get my-cluster --admin
   ```

2. Ändern Sie den Kontext in den vSphere-Namespace, in dem der TKG-Zielcluster bereitgestellt wird.

   ```
   kubectl config use-context my-cluster-admin@my-cluster
   ```

3 Wenn der Cluster über kein Paket-Repository verfügt, z. B. das tanzu-standard-Repository, installieren Sie ein Repository.

   Sie können diesen Schritt überspringen, wenn es sich bei Ihrem Zielcluster um einen planbasierten Legacy-Cluster handelt. Für planbasierte Cluster wird das tanzu-standard-Paket-Repository automatisch im tanzu-package-repo-global-Namespace aktiviert.

   ```
   tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --namespace tkg-system
   ```

   - PACKAGE-REPO-NAME ist der Name des Paket-Repositorys, z. B. des tanzu-standard-Repositorys, oder der Name einer privaten Image-Registrierung, die mit ADDITIONAL_IMAGE_REGISTRY-Variablen konfiguriert ist.
   - PACKAGE-REPO-ENDPOINT ist die URL des Paket-Repositorys.

4 Falls noch nicht geschehen, installieren Sie cert-manager.

   Weitere Informationen finden Sie unter Installieren des Zertifikatmanagers.

Ergebnisse

Sie können jetzt den externen Webhook für CSI-Snapshots bereitstellen.

Bereitstellen eines externen CSI-Snapshot-Webhooks

Führen Sie die folgenden Schritte aus, um den externen Webhook für CSI-Snapshots auf einem TKG-Dienst-Cluster bereitzustellen.

Verfahren

1 Bestätigen Sie, dass das externe Webhook-Paket für CSI-Snapshots im Cluster verfügbar ist.

   ```
   tanzu package available list -A
   ```

   Wenn das Paket nicht verfügbar ist, stellen Sie sicher, dass das Paket-Repository, das das erforderliche externe CSI-Snapshot-Webhook-Paket enthält, ordnungsgemäß installiert ist. Anweisungen finden Sie in Schritt 3 unter Vorbereiten eines TKG-Dienst-Clusters für die Installation des externen Webhooks für CSI-Snapshots.

2 Rufen Sie die Version des verfügbaren Pakets ab.

   ```
   tanzu package available list external-csi-snapshot-webhook.tanzu.vmware.com -A
   ```

3 Installieren Sie das Paket mit der entsprechenden verfügbaren Version.

   ```
   tanzu package install external-csi-snapshot-webhook --package external-csi-snapshot-webhook.tanzu.vmware.com --version AVAILABLE-PACKAGE-VERSION --namespace kube-system
   ```

   AVAILABLE-PACKAGE-VERSION gibt die Paketversion an, die Sie in Schritt 2 erhalten haben.

4 Bestätigen Sie, dass das externe Webhook-Paket für CSI-Snapshots installiert wurde.

```
tanzu package installed list -A
```

Um weitere Details zum Paket anzuzeigen, können Sie auch den folgenden Befehl ausführen:

```
tanzu package installed get external-csi-snapshot-webhook --namespace kube-system
```

5 Vergewissern Sie sich, dass die external-csi-snapshot-webhook-App erfolgreich in Ihrem TARGET-NAMESPACE abgeglichen wurde.

```
kubectl get apps -A
```

Wenn der Status nicht `Reconcile Succeeded` lautet, zeigen Sie die vollständigen Statusdetails der external-csi-snapshot-webhook-App an. Das Anzeigen des vollständigen Status kann Ihnen bei der Behebung des Problems helfen.

```
kubectl get app external-csi-snapshot-webhook --namespace kube-system -o yaml
```

Falls Sie das Problem nicht beheben können, deinstallieren Sie das Paket mit dem folgenden Befehl, bevor Sie es erneut installieren.

```
tanzu package installed delete external-csi-snapshot-webhook --namespace kube-system
```

6 Bestätigen Sie, dass external-csi-snapshot-webhook ausgeführt wird, indem Sie alle Pods im Cluster auflisten.

```
kubectl get pods -A
```

Stellen Sie sicher, dass external-csi-snapshot-webhook-Pods im kube-system-Namespace erstellt werden.

Installieren und Bereitstellen des vSphere PVCSI Webhooks

Installieren Sie den vSphere-PVCSI-Webhook auf einem TKG-Dienst-Cluster, und stellen Sie ihn bereit. Der vSphere PVCSI-Webhook ist eine Komponente mit Rückruf, die auf CSI-Zugangsanforderungen reagiert. Er ist für die Validierung von Kubernetes-Objekten wie Beanspruchungen dauerhafter Volumes, dauerhafter Volumes, Speicherklassen usw. verantwortlich.

Der vSphere-PVCSI-Webhook wird automatisch in Supervisor installiert. Dieses Thema gilt nur für TKG-Dienst-Cluster.

Voraussetzungen

- Sie verfügen über einen ausgeführten Supervisor.
- Tanzu-CLI und kubectl sind installiert. Weitere Informationen finden Sie unter Installieren von CLI-Tools für TKG-Dienst-Cluster.

- Sie haben sich beim TKG-Cluster angemeldet. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu TKG-Dienst-Clustern mithilfe der vCenter SSO-Authentifizierung.

Vorbereiten eines TKG-Dienst-Clusters für die Installation des vSphere PVCSI-Webhooks

Führen Sie die folgenden Schritte aus, um den TKG-Dienst-Cluster für die Installation des vSphere PVCSI-Webhooks vorzubereiten.

Verfahren

1. Rufen Sie die Admin-Anmeldedaten des TKG-Clusters ab, in dem Sie den vSphere-pvCSI-Webhook bereitstellen möchten.

   ```
   tanzu cluster kubeconfig get my-cluster --admin
   ```

2. Ändern Sie den Kontext in den vSphere-Namespace, in dem der TKG-Zielcluster bereitgestellt wird.

   ```
   kubectl config use-context my-cluster-admin@my-cluster
   ```

3. Wenn der Cluster über kein Paket-Repository mit installiertem vsphere-pv-csi-webhook-Paket verfügt, z. B. das tanzu-standard-Repository, installieren Sie ein Repository.

   Sie können diesen Schritt überspringen, wenn es sich bei Ihrem Zielcluster um einen planbasierten Legacy-Cluster handelt. Für planbasierte Cluster wird das tanzu-standard-Paket-Repository automatisch im tanzu-package-repo-global-Namespace aktiviert.

   ```
   tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --namespace tkg-system
   ```

   - PACKAGE-REPO-NAME ist der Name des Paket-Repositorys, z. B. des tanzu-standard-Repositorys, oder der Name einer privaten Image-Registrierung, die mit ADDITIONAL_IMAGE_REGISTRY-Variablen konfiguriert ist.
   - PACKAGE-REPO-ENDPOINT ist die URL des Paket-Repositorys.

4. Falls noch nicht geschehen, installieren Sie cert-manager.

   Weitere Informationen finden Sie unter Installieren des Zertifikatmanagers.

Ergebnisse

Sie können jetzt den vSphere PVCSI-Webhook bereitstellen.

Bereitstellen eines vSphere PVCSI-Webhooks

Führen Sie die folgenden Schritte aus, um den vSphere-PVCSI-Webhook auf einem TKG-Dienst-Cluster bereitzustellen.

Verfahren

1 Bestätigen Sie, dass das vSphere-pvCSI-Webhook-Paket im Cluster verfügbar ist.

```
tanzu package available list -A
```

Wenn das Paket nicht verfügbar ist, stellen Sie sicher, dass das Paket-Repository, das die erforderlichen vSphere PVCSI-Webhook-Paket enthält, ordnungsgemäß installiert ist. Anweisungen finden Sie in Schritt 3 unter Vorbereiten eines TKG-Dienst-Clusters für die Installation des vSphere PVCSI-Webhooks.

2 Rufen Sie die Version des verfügbaren Pakets ab.

```
tanzu package available list vsphere-pv-csi-webhook.tanzu.vmware.com -A
```

3 Installieren Sie das Paket mit der entsprechenden verfügbaren Version.

```
tanzu package install vsphere-pv-csi-webhook --package vsphere-pv-csi-
webhook.tanzu.vmware.com --version AVAILABLE-PACKAGE-VERSION --namespace TARGET-NAMESPACE
```

- TARGET-NAMESPACE gibt den Namespace an, in dem das vsphere-pv-csi-webhook-Paket installiert werden soll.

  Hinweis TARGET-NAMESPACE muss mit dem Namespace identisch sein, in dem das vsphere-pv-csi-Paket installiert ist.

  Wenn Sie das `--namespace`-Flag nicht angeben, installiert die Tanzu-CLI das Paket und die zugehörigen Ressourcen im Standard-Namespace, z. B. `vmware-system-csi` für das vsphere-pv-csi-webhook-Paket. Der angegebene Namespace muss bereits vorhanden sein, z. B. durch die Ausführung von `kubectl create namespace vmware-system-csi`.

- AVAILABLE-PACKAGE-VERSION gibt die Paketversion an, die Sie in Schritt 2 erhalten haben.

4 Bestätigen Sie, dass das vSphere PVCSI-Webhook-Paket installiert wurde.

```
tanzu package installed list -A
```

Um weitere Details zum Paket anzuzeigen, können Sie auch den folgenden Befehl ausführen:

```
tanzu package installed get vsphere-pv-csi-webhook --namespace TARGET-NAMESPACE
```

5 Vergewissern Sie sich, dass die vsphere-pv-csi-webhook-App erfolgreich in Ihrem TARGET-NAMESPACE abgeglichen wurde.

```
kubectl get apps -A
```

Wenn der Status nicht `Reconcile Succeeded` lautet, zeigen Sie die vollständigen Statusdetails der vsphere-pv-csi-webhook-App an. Das Anzeigen des vollständigen Status kann Ihnen bei der Behebung des Problems helfen.

```
kubectl get app vsphere-pv-csi-webhook --namespace TARGET-NAMESPACE -o yaml
```

Falls Sie das Problem nicht beheben können, deinstallieren Sie das Paket mit dem folgenden Befehl, bevor Sie es erneut installieren.

```
tanzu package installed delete vsphere-pv-csi-webhook --namespace TARGET-NAMESPACE
```

6 Bestätigen Sie, dass vsphere-pv-csi-webhook ausgeführt wird, indem Sie alle Pods im Cluster auflisten.

```
kubectl get pods -A
```

Stellen Sie sicher, dass vsphere-pv-csi-webhook-Pods in vmware-system-csi ODER im TARGET-NAMESPACE erstellt werden.

Erstellen von Snapshots in einem TKG-Dienst-Cluster

Sie können einen Snapshot dynamisch bereitstellen oder einen vorab bereitgestellten Volume-Snapshot eines Block-Volumes in einem TKG-Dienst-Cluster erstellen. Sie können auch einen vorhandenen Snapshot wiederherstellen.

Hinweis Sie sollten keine Volume-Snapshot-Klassen erstellen und diese zum Erstellen von Volume-Snapshots verwenden. Verwenden Sie nur eine bereits vorhandene Volume-Snapshot-Klasse. Bereits vorhandene Volume-Snapshot-Klassen unterstützen nur die deletionPolicy „Delete“ (Löschen). Das Erstellen von Volume-Snapshot-Klassen, bei denen deletionPolicy auf „Retain“ (Beibehalten) festgelegt ist, wird nicht unterstützt.

Erstellen eines dynamisch bereitgestellten Snapshots in einem TKG-Dienst-Cluster

Stellen Sie dynamisch einen Snapshot in einem TKG-Dienst-Cluster bereit.

Verfahren

1 Stellen Sie sicher, dass die StorageClass vorhanden ist.

```
$ kubectl get sc
NAME                              PROVISIONER              RECLAIMPOLICY
VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
gc-storage-profile                csi.vsphere.vmware.com   Delete
Immediate              true                   11d
gc-storage-profile-latebinding    csi.vsphere.vmware.com   Delete
WaitForFirstConsumer   true                   11d
```

2 Erstellen Sie eine PVC mithilfe der StorageClass aus Schritt 1.

Verwenden Sie die folgende YAML als Beispiel.

```
$ cat example-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-raw-block-pvc
spec:
  volumeMode: Block
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: gc-storage-profile
```

```
$ kubectl apply -f example-pvc.yaml
```

```
$ kubectl get pvc
NAME                    STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS         AGE
example-raw-block-pvc   Bound    pvc-4c0c030d-25ac-4520-9a04-7aa9361dfcfc   1Gi
RWO            gc-storage-profile   2m1s
```

3 Stellen Sie sicher, dass die VolumeSnapshotClass verfügbar ist.

```
$ kubectl get volumesnapshotclass
NAME                          DRIVER                    DELETIONPOLICY   AGE
volumesnapshotclass-delete   csi.vsphere.vmware.com   Delete           11d
```

4 Erstellen Sie einen VolumeSnapshot mithilfe der VolumeSnapshotClass, die Sie in Schritt 3 abgerufen haben.

```
$cat example-snapshot.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: example-raw-block-snapshot
spec:
  volumeSnapshotClassName: volumesnapshotclass-delete
  source:
    persistentVolumeClaimName: example-raw-block-pvc
```

```
$ kubectl apply -f example-snapshot.yaml
```

```
$ kubectl get volumesnapshot
NAME                          READYTOUSE   SOURCEPVC               SOURCESNAPSHOTCONTENT
RESTORESIZE   SNAPSHOTCLASS                SNAPSHOTCONTENT
                             CREATIONTIME   AGE
```

```
example-raw-block-snapshot   true         example-raw-block-pvc
1Gi           volumesnapshotclass-delete   snapcontent-ae019c16-
b07c-4a92-868b-029babd641d3   6s             6s
```

Erstellen eines vorab bereitgestellten Snapshots in einem TKG-Dienst-Cluster

Erstellen Sie einen vorab bereitgestellten Volume-Snapshot eines beliebigen Block-Volumes (PVC) in einem TKG-Dienst-Cluster. Verwenden Sie dazu einen Volume-Snapshot desselben Block-Volumes (PVC) aus dem Supervisor.

Führen Sie die folgenden Schritte aus, um einen Volume-Snapshot in einem neuen TKG-Cluster statisch zu erstellen, indem Sie die Informationen aus dem verbliebenen zugrunde liegenden Snapshot in Supervisor verwenden.

Hinweis Volume-Snapshots können nicht direkt auf einem Supervisor erstellt werden.

Voraussetzungen

- Machen Sie sich mit den Informationen zum Erstellen von Kubernetes-Snapshots vertraut. Weitere Informationen finden Sie auf der Seite zu Volume-Snapshots in der Kubernetes-Dokumentation.
- Stellen Sie sicher, dass der Volume-Snapshot die folgenden Bedingungen erfüllt:
  - Der Volume-Snapshot befindet sich im selben Namespace, in dem sich die Quell-PVC befindet.
  - Der Volume-Snapshot befindet sich im selben Namespace, in dem sich der TKG-Cluster befindet.

Sie können auch einen Volume-Snapshot, der von einem anderen TKG-Cluster im selben Namespace nicht mehr benötigt wird, in einem neuen TKG-Cluster wiederverwenden. Ändern Sie dazu die `deletionPolicy` des `VolumeSnapshotContent` im ursprünglichen TKG-Cluster in `Retain` und löschen Sie dann die entsprechenden `VolumeSnapshot`- und `VolumeSnapshotContent`-Objekte.

Verfahren

1 Notieren Sie sich den Namen des ursprünglichen `VolumeSnapshot`-Objekts im Supervisor.

Wenn Sie den Volume-Snapshot aus einem alten TKG-Cluster wiederverwenden, können Sie auch den `VolumeSnapshot`-Namen aus Supervisor aus dem `snapshotHandle` des alten `VolumeSnapshotContent`-Objekts im alten TKG-Cluster abrufen.

2 Erstellen Sie ein `VolumeSnapshotContent`-Objekt.

Geben Sie in der YAML-Datei den Wert des folgenden Elements an:

Geben Sie für `snapshotHandle` den `VolumeSnapshot`-Namen aus Supervisor ein, den Sie in Schritt 1 abgerufen haben.

Hinweis Hinweis: Wenn Sie einen Volume-Snapshot aus einem anderen TKG-Cluster wiederverwenden, löschen Sie die `VolumeSnapshot`- und `VolumeSnapshotContent`-Objekte aus dem alten TKG-Cluster (legen Sie dabei `deletionPolicy` zum Beibehalten auf „Retain“ fest), bevor Sie einen `VolumeSnapshotContent` im neuen TKG-Cluster erstellen.

Verwenden Sie das folgende YAML-Manifest als Beispiel.

```
--------
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
 name: static-tkg-block-snapshotcontent
spec:
 deletionPolicy: Delete
 driver: csi.vsphere.vmware.com
 source:
  snapshotHandle: "supervisor-block-volumeSnapshot-name" # Enter the VolumeSnapshot name
from the Supervisor.
 volumeSnapshotRef:
  name: static-tkg-block-snapshot
  namespace: "supervisor-tkg-namespace" # Enter the namespace of Tanzu Kubernetes Grid
Cluster.
---------
```

3 Erstellen Sie einen `VolumeSnapshot`, der dem in Schritt 2 erstellten `VolumeSnapshotContent`-Objekt entspricht.

```
---------
apiVersion: snapshot.storage.k8s.io/vl
kind: VolumeSnapshot
metadata:
 name: static-tkg-block-snapshot
spec:
 source:
  volumeSnapshotContentName: static-tkg-block-snapshotcontent
---------
```

4 Für den `VolumeSnapshot`, den Sie in Schritt 3 erstellt haben, muss ReadyToUse als „true“ markiert sein.

```
kubecti getvolumesnapshot static-tkg-block-snapshot
NAME                        READYTOUSE SOURCEPVC SOURCESNAPSHOTCONTENT
RESTORESIZE SNAPSHOTCLASS SNAPSHOTCONTENT                  CREATIONTIME AGE
static-tkg-block-snapshot  true                static-tkg-biock-snapshotcontent
5Gi                       static-tkg-block-snapsnotcontent 76m          22m
```

Wiederherstellen eines Volume-Snapshots in einem TKG-Dienst-Cluster

Stellen Sie einen bereits erstellten Volume-Snapshot wieder her.

Verfahren

1 Überprüfen Sie, ob der wiederherzustellende Volume-Snapshot im TKG-Cluster verfügbar ist.

```
$ kubectl get volumesnapshot
NAME                          READYTOUSE   SOURCEPVC
SOURCESNAPSHOTCONTENT   RESTORESIZE   SNAPSHOTCLASS
SNAPSHOTCONTENT                                    CREATIONTIME   AGE
example-raw-block-snapshot   true         example-raw-block-pvc
1Gi           volumesnapshotclass-delete   snapcontent-ae019c16-
b07c-4a92-868b-029babd641d3   2m36s          2m36s
```

2 Erstellen Sie eine PVC aus dem Volume-Snapshot.

```
$ cat example-restore.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-raw-block-restore
spec:
  storageClassName: gc-storage-profile
  dataSource:
    name: example-raw-block-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
  volumeMode: Block
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

```
$ kubectl apply -f example-restore.yaml
```

```
$ kubectl get pvc
NAME                          STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS         AGE
example-raw-block-pvc       Bound    pvc-4c0c030d-25ac-4520-9a04-7aa9361dfcfc   1Gi
RWO            gc-storage-profile   11m
example-raw-block-restore   Bound    pvc-96eaab16-9ec1-446a-9392-e86d13c9b2e2   1Gi
RWO            gc-storage-profile   2m8s
```

Verwalten des Speichers für TKG-Dienst-Cluster

16

Dieser Abschnitt enthält Informationen zum Verwalten des Speichers für TKG-Dienst-Cluster.

Lesen Sie als Nächstes die folgenden Themen:

- Speicherkonzepte für TKG-Dienstcluster
- Überlegungen zur Verwendung von Knoten-Volume-Bereitstellungen
- Erstellen einer vSphere-Speicherrichtlinie für TKG-Dienst-Cluster
- Bereitstellen eines dynamischen dauerhaften Volumes für eine statusbehaftete Anwendung in einem TKG-Dienst-Cluster
- Bereitstellen eines statischen persistenten Volumes in einem TKG-Dienst-Cluster
- Erweiterung persistenter Volumes für TKG-Dienst-Cluster

Speicherkonzepte für TKG-Dienstcluster

TKG-Clusterarbeitslasten erfordern möglicherweise persistenten Speicher. Informationen zu vSphere-Speicherkonzepten und Überlegungen zu TKG-Dienst-Clustern finden Sie in diesem Thema.

vSphere-Speicherrichtlinien für TKG-Dienst-Cluster

Zum Bereitstellen persistenter Speicherressourcen für die TKG-Dienst-Cluster konfiguriert ein vSphere-Administrator vSphere-Speicherrichtlinien, die unterschiedliche Speicheranforderungen abdecken. Anschließend fügt der Administrator dem vSphere-Namespace, in dem TKG-Cluster bereitgestellt werden, eine oder mehrere Speicherrichtlinien hinzu. Speicherrichtlinien, die einem vSphere-Namespace zugewiesen sind, bestimmen die Platzierung von TKG-Clusterknoten und -Arbeitslasten in der vSphere-Speicherumgebung. Außerdem bestimmen sie, auf welche Datenspeicher TKG-Cluster zugreifen und diese für den persistenten Speicher verwenden können.

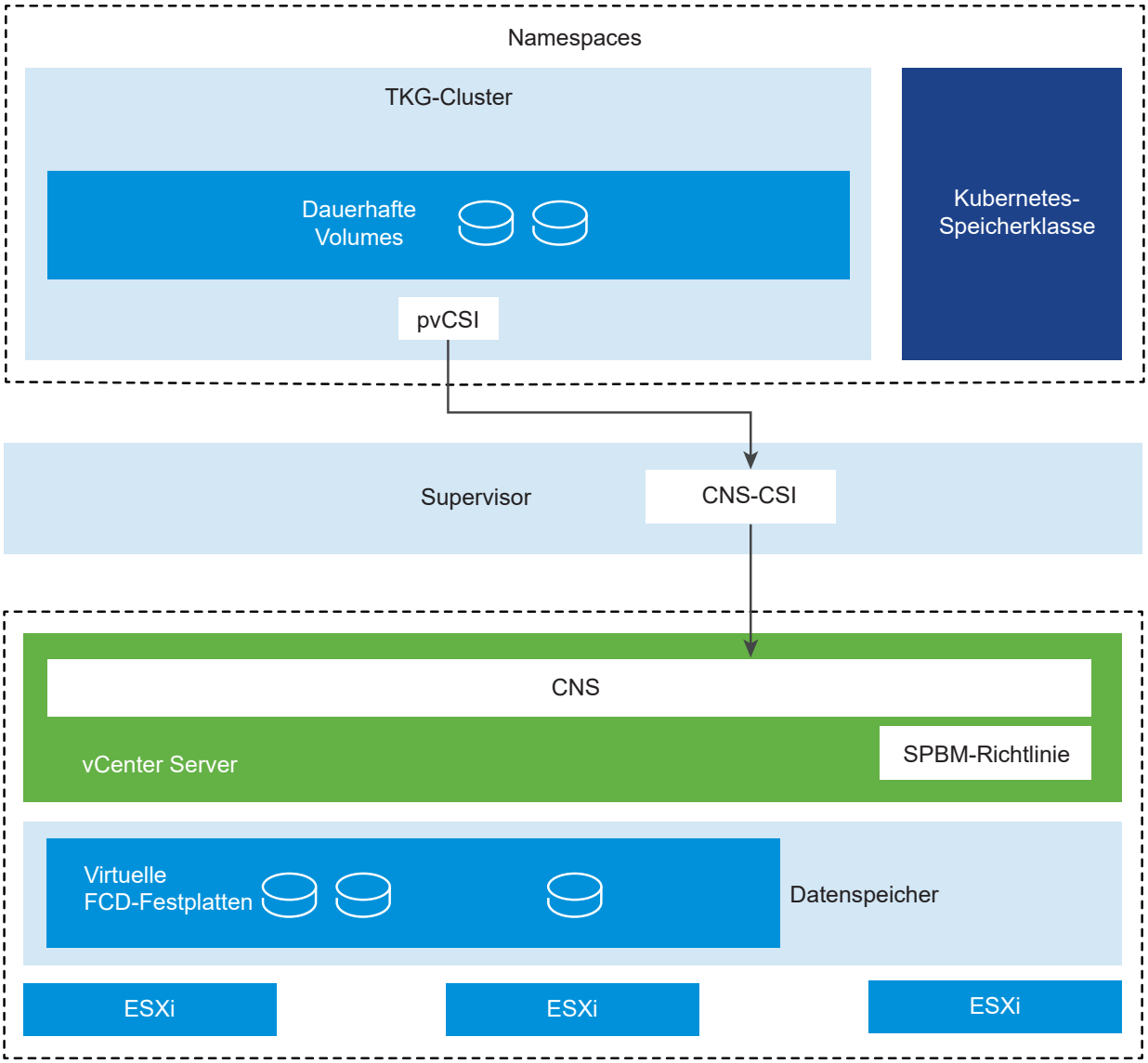
Wenn einem vSphere-Namespace eine vSphere-Speicherrichtlinie zugewiesen wird, erstellt das System übereinstimmende Kubernetes-Speicherklassen für diesen vSphere-Namespace. Die übereinstimmenden Kubernetes-Speicherklassen werden auch an die TKG-Cluster weitergegeben, die in diesem vSphere-Namespacebereitgestellt werden.

In einem TKG-Cluster weist jede Speicherklasse zwei Editionen auf: eine mit dem `Immediate`- und die andere mit dem `WaitForFirstConsumer`-Bindungsmodus. Welche Edition Sie auswählen, richtet sich nach Ihren Anforderungen. Weitere Informationen hierzu finden Sie unter Speicherklasseneditionen für TKG-Dienstcluster.

Integrieren von TKG-Dienst-Clustern in vSphere-Speicher

Für die Integration in Supervisor und in den vSphere-Speicher verwenden TKG-Cluster die paravirtuelle CSI (pvCSI).

Die pvCSI ist die für TKG-Cluster geänderte Version des vSphere CNS-CSI-Treibers. Die pvCSI-Komponente befindet sich im TKG-Cluster und ist für alle aus dem TKG-Cluster stammenden, speicherbezogenen Anforderungen zuständig. Die Anforderungen werden an CNS-CSI übermittelt und von dort an CNS in vCenter Server weitergeleitet. Dies führt dazu, dass pvCSI nicht direkt mit der CNS-Komponente kommuniziert, sondern bei allen Speicherbereitstellungsvorgängen auf CNS-CSI angewiesen ist. Im Gegensatz zu CNS-CSI benötigt pvCSI keine Anmeldedaten für die Infrastruktur. pvCSI ist mit einem Dienstkonto im vSphere-Namespace konfiguriert.

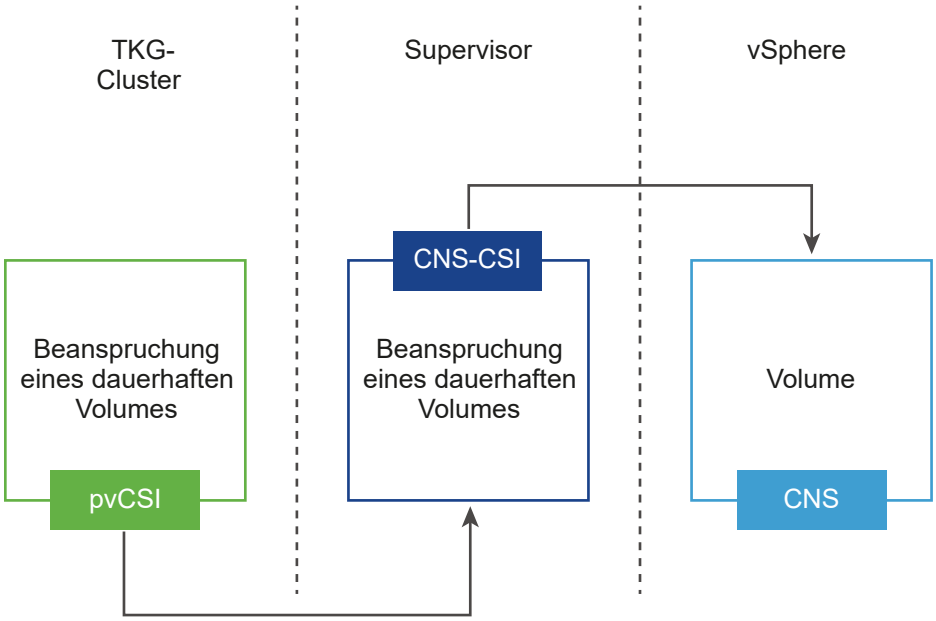


Erstellen eines persistenten Volumes

Das Diagramm veranschaulicht, wie verschiedene Komponenten für speicherbezogene Vorgänge innerhalb eines TKG-Clusters interagieren, etwa beim Erstellen einer Beanspruchung eines dauerhaften Volumes (Persistent Volume Claim, PVC).

Der DevOps-Ingenieur erstellt eine PVC mithilfe von „kubectl“ im TKG-Cluster. Diese Aktion generiert eine übereinstimmende PVC in Supervisor und löst „CNS-CSI“ aus, wodurch die CNS API zum Erstellen von Volumes aufgerufen wird.

Nach erfolgreicher Erstellung eines Volumes wird der Vorgang über Supervisor wieder an den TKG-Cluster zurückgegeben. Die Clusterbenutzer können das persistente Volume und die Beanspruchung eines dauerhaften Volumes im gebundenen Zustand im Supervisor sehen. Außerdem sehen sie das persistente Volume und die Beanspruchung eines dauerhaften Volumes im gebundenen Zustand auch im TKG-Cluster.



Überlegungen zur Verwendung von Knoten-Volume-Bereitstellungen

Sie können einen TKG-Dienstcluster mit einem oder mehreren Knoten-Volume-Bereitstellungen bereitstellen. Bevor Sie dies tun, beachten Sie wichtige Überlegungen.

Überlegungen zu Knoten-Volume-Bereitstellungen

In der Tabelle werden Bereitstellungspunkte für Knoten-Volumes für TKG-Dienstcluster zusammengefasst. Vollständige Informationen zu Einschränkungen bei Knoten-Volume-Bereitstellungen finden Sie im folgenden KB-Artikel: https://kb.vmware.com/s/article/92153.

Informationen zum Ändern einer Volume-Bereitstellung nach der Bereitstellung eines Clusters finden Sie auch im folgenden Thema: Manuelles Skalieren eines Clusters mithilfe von „Kubectl“.

Volume-Mount	Knoten	Support	Abst
`/var/lib/containerd`	Worker	Voll	Erhöht die für die Zwischenspeicherung von Container-Images verfügbare Größe.
`/var/lib/kubelet`	Worker	Voll	Erhöht die verfügbare Größe für flüchtigen Container-Speicher.

Volume-Mount	Knoten	Support	Abst
`/var/lib/etcd`	Steuerungsebene	Keine	Erhöht die Größe von etcd nicht, da etcd einen harten Grenzwert von 2 GB aufweist. Kann auch die Cluster-Erstellung aufgrund von PVC-Zeitüberschreitungen verhindern.

`/` (root)
`/var`
`/var/lib`
`/etc`
	Steuerungsebene
	Worker
		Keine	Alle Verzeichnisse, die von Kernsystemprozessen verwendet werden, werden nicht unterstützt, einschließlich der aufgeführten und anderer Verzeichnisse (Liste ist nicht erschöpfend).

Erstellen einer vSphere-Speicherrichtlinie für TKG-Dienst-Cluster

Eine vSphere-Speicherrichtlinie, die Sie einem vSphere-Namespace zuweisen, wird in eine Kubernetes-Speicherklasse konvertiert. Sie verwenden diese Speicherklasse, um zu steuern, wie TKG-Clusterknoten und dauerhafte Volumes in vSphere-Datenspeichern platziert werden. Eine vSphere-Speicherrichtlinie für vSphere-Zonen muss mit dem Speicher in allen vSphere-Clustern kompatibel sein, die die Zonentopologie umfassen.

Definieren Sie eine vSphere-Speicherrichtlinie für einen Einzelzonen-Supervisor.

Führen Sie die Schritte zum Erstellen einer Speicherklasse für einen Einzelzonen-Supervisor aus.

1 Wählen Sie mithilfe des vSphere Client **Richtlinien und Profile** aus.

2 Wählen Sie **VM-Speicherrichtlinien > Erstellen** aus.

3 Aktivieren Sie den vCenter Server.

4 Geben Sie einen beschreibenden Namen für die Speicherrichtlinie ein, z. B. *TKG2-cluster-storage-class*.

   Derselbe Name wird für die erstellte Speicherklasse verwendet.

5 Wählen Sie für Speicher **Richtlinienstruktur** die Option **Regeln für Speicher „VMFS“ aktivieren** aus.

6 Wählen Sie als VMFS-Regeln **Speicherplatz reservieren, wenn möglich** aus.

7 Überprüfen Sie die Einstellungen für die Speicherkompatibilität und klicken Sie auf **Weiter**.

8 Klicken Sie auf **Beenden**, um die Erstellung der Speicherrichtlinie abzuschließen.

Informationen zum Zuweisen der Speicherrichtlinie zu einem vSphere-Namespace finden Sie unter Konfigurieren eines vSphere-Namespace für TKG-Dienst-Cluster.

Definieren Sie eine vSphere-Speicherrichtlinie für einen Drei-Zonen-Supervisor

Wenn Sie Supervisor in drei Zonen bereitstellen, führen Sie die gleichen Schritte wie oben beschrieben aus, mit den folgenden Ergänzungen:

- Wählen Sie als **Richtlinienstruktur** für den Speicher **Speichertopologie > Verbrauchdomäne aktivieren** aus.
- Geben Sie für **Verbrauchsdomäne** als Speicherrichtlinientyp **Zonal** an.

Abbildung 16-1. Zonale Speicherklasse 1 von 2

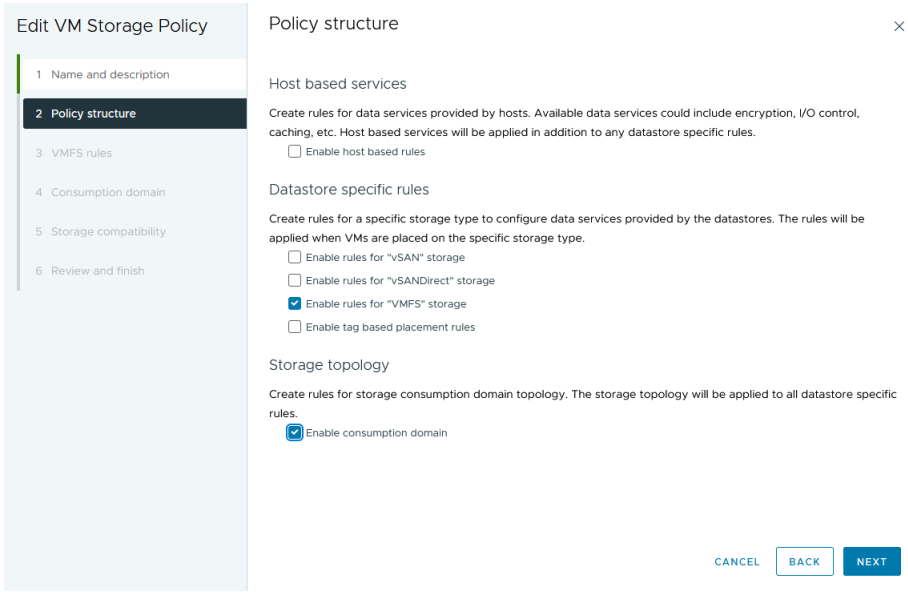
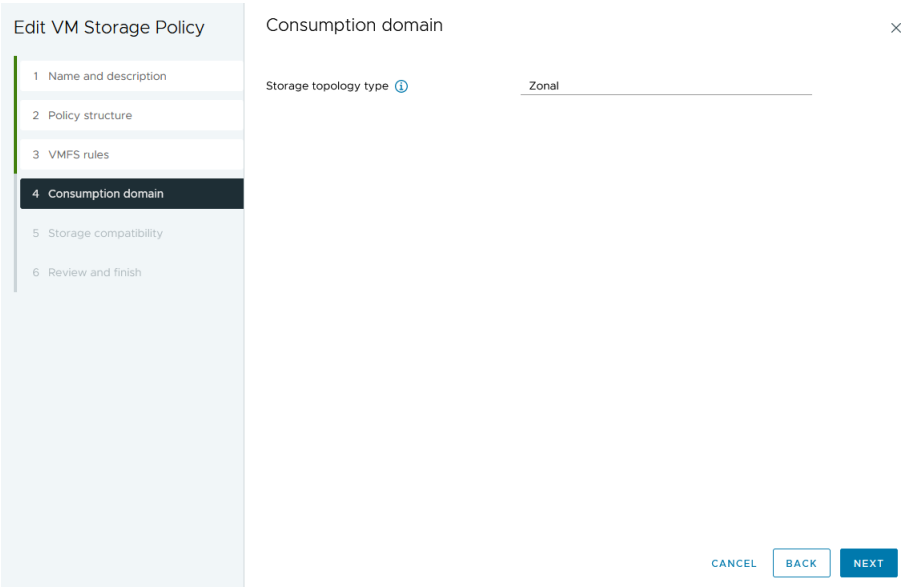


Abbildung 16-2. Zonale Speicherklasse 2 von 2



Erweiterte Erstellung von vSphere-Speicherrichtlinien

Je nach Umgebung benötigen Sie möglicherweise zusätzliche Einstellungen für vSAN oder vVols. Wenn Sie VMFS und NFS verwenden, enthält die vSphere-Speicherrichtlinie Tags.

Informationen zum Erstellen erweiterter Speicherrichtlinientypen, wie z. B. Tag-basierter Speicher, die auf einen vSphere-Namespace zur Verwendung mit TKG-Clustern angewendet werden können, finden Sie unter *Installieren und Konfigurieren der vSphere IaaS-Steuerungsebene*.

Bereitstellen eines dynamischen dauerhaften Volumes für eine statusbehaftete Anwendung in einem TKG-Dienst-Cluster

Statusbehaftete Anwendungen, wie z. B. Datenbanken, speichern Daten zwischen Sitzungen und benötigen dauerhaften Speicher zum Speichern der Daten. Die gespeicherten Daten werden als Anwendungsstatus bezeichnet. Sie können die Daten später abrufen und in der nächsten Sitzung verwenden. Kubernetes stellt dauerhafte Volumes als Objekte bereit, die ihren Status und ihre Daten beibehalten können.

In der vSphere-Umgebung werden die Objekte eines dauerhaften Volumes von virtuellen Festplatten gesichert, die sich in Datenspeichern befinden. Datenspeicher werden durch Speicherrichtlinien dargestellt. Nachdem der vSphere-Administrator eine Speicherrichtlinie (z. B. **`Gold`**) erstellt und dem Namespace in einem Supervisor zugewiesen hat, wird die Speicherrichtlinie als übereinstimmende Kubernetes-Speicherklasse im vSphere-Namespace und in allen verfügbaren TKG-Clustern angezeigt.

Als DevOps-Ingenieur können Sie die Speicherklasse in Ihren Anspruchsspezifikationen für dauerhafte Volumes verwenden. Anschließend können Sie eine Anwendung bereitstellen, die Speicher aus dem Anspruch für dauerhafte Volumes verwendet. In diesem Beispiel wird das dauerhafte Volume für die Anwendung dynamisch erstellt.

Voraussetzungen

Stellen Sie sicher, dass Ihr vSphere-Administrator eine geeignete Speicherrichtlinie erstellt und dem Namespace die Richtlinie zugewiesen hat.

Verfahren

1. Greifen Sie in der vSphere Kubernetes-Umgebung auf Ihren Namespace zu.
2. Stellen Sie sicher, dass die Speicherklassen verfügbar sind.

3 Erstellen Sie einen Anspruch für dauerhafte Volumes.

a Erstellen Sie eine YAML-Datei, die die Konfiguration der Beanspruchung eines dauerhaften Volumes enthält.

In diesem Beispiel verweist die Datei auf die Speicherklasse **`Gold`**.

Um ein persistentes Volume im ReadWriteMany-Modus bereitzustellen, setzen Sie `accessModes` auf `ReadWriteMany`.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gold
  resources:
    requests:
        storage: 3Gi
```

b Wenden Sie die Beanspruchung eines dauerhaften Volumes auf den TKG-Cluster an.

```
kubectl apply -f pvc_name.yaml
```

Dieser Befehl erstellt dynamisch ein dauerhaftes Kubernetes-Volume sowie ein vSphere-Volume mit einer zugrunde liegenden virtuellen Festplatte, die die Speicheranforderungen der Beanspruchung erfüllt.

c Überprüfen Sie den Status der Beanspruchung eines dauerhaften Volumes.

```
kubectl get pvc my-pvc
```

Die Ausgabe zeigt, dass das Volume an Anspruch für dauerhafte Volumes gebunden ist.

```
NAME     STATUS    VOLUME   CAPACITY   ACCESSMODES   STORAGECLASS   AGE
my-pvc   Bound     my-pvc   2Gi        RWO           gold           30s
```

4 Erstellen Sie einen Pod, der das dauerhafte Volume mountet.

a Erstellen Sie eine YAML-Datei, die das dauerhafte Volume enthält.

Die Datei enthält diese Parameter.

```
...
volumes:
    - name: my-pvc
      persistentVolumeClaim:
        claimName: my-pvc
```

b Stellen Sie den Pod aus der YAML-Datei bereit:

```
kubectl create -f pv_pod_name.yaml
```

c Stellen Sie sicher, dass der Pod erstellt wurde.

```
kubectl get pod
```

```
NAME       READY   STATUS    RESTARTS   AGE
pod_name   1/1     Ready     0          40s
```

Ergebnisse

Der von Ihnen konfigurierte Pod verwendet dauerhaften Speicher, der im Anspruch für dauerhafte Volumes beschrieben wird.

Bereitstellen eines statischen persistenten Volumes in einem TKG-Dienst-Cluster

Sie können ein Block-Volume statisch in einem TKG-Dienst-Cluster mithilfe einer nicht verwendeten Beanspruchung eines persistenten Volumes (Persistent Volume Claim, PVC) im Supervisor erstellen.

Die PVC muss die folgenden Bedingungen erfüllen:

- Die PVC ist in demselben Namespace vorhanden, in dem sich der TKG-Cluster befindet.
- Die PVC ist noch nicht an einen Pod in einem TKG-Cluster oder an einen vSphere Pod in Supervisor angehängt.

Mit der statischen Bereitstellung können Sie eine PVC, die von einem anderen TKG-Cluster nicht mehr benötigt wird, auch in einem neuen TKG-Cluster wiederverwenden. Ändern Sie dazu die `Reclaim policy` des persistenten Volumes (PV) im ursprünglichen TKG-Cluster in `Retain`, und löschen Sie dann die entsprechende PVC.

Führen Sie die folgenden Schritte aus, um eine PVC in einem neuen TKG-Cluster statisch zu erstellen. Verwenden Sie dazu die Informationen aus dem übrig gebliebenen zugrunde liegenden Volume.

Verfahren

1 Notieren Sie sich den Namen der ursprünglichen PVC im Supervisor.

Wenn Sie die PVC von einem alten TKG-Cluster wiederverwenden, können Sie den PVC-Namen des `volumeHandle` des alten PV-Objekts im TKG-Cluster abrufen.

2 Erstellen Sie ein PV.

Geben Sie in der YAML-Datei die Werte der folgenden Elemente an:

- Für `storageClassName` können Sie den Namen der Speicherklasse eingeben, der von Ihrer PVC im Supervisor verwendet wird.
- Geben Sie für `volumeHandle` den PVC-Namen ein, den Sie abgerufen haben.

Wenn Sie ein Volume von einem anderen TKG-Cluster wiederverwenden, löschen Sie die PVC- und PV-Objekte aus dem alten TKG-Cluster, bevor Sie ein PV im neuen TKG-Cluster erstellen.

Verwenden Sie das folgende YAML-Manifest als Beispiel.

```
apiVersion: v1
   kind: PersistentVolume
   metadata:
     name: static-tkg-block-pv
     annotations:
       pv.kubernetes.io/provisioned-by: csi.vsphere.vmware.com
   spec:
     storageClassName: gc-storage-profile
     capacity:
       storage: 2Gi
     accessModes:
       - ReadWriteOnce
     persistentVolumeReclaimPolicy: Delete
     claimRef:
       namespace: default
       name: static-tkg-block-pvc
     csi:
       driver: "csi.vsphere.vmware.com"
       volumeAttributes:
         type: "vSphere CNS Block Volume"
       volumeHandle: "supervisor-block-pvc-name"  # Enter the PVC name from the Supervisor.
```

3 Erstellen Sie eine PVC, die dem von Ihnen erstellten PV-Objekt entspricht.

Legen Sie für `storageClassName` denselben Wert wie im PV fest.

```
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: static-tkg-block-pvc
   spec:
     accessModes:
       - ReadWriteOnce
     resources:
```

```
      requests:
        storage: 2Gi
    storageClassName: gc-storage-profile
    volumeName: static-tkg-block-pv
```

4 Stellen Sie sicher, dass die PVC an das von Ihnen erstellte PV gebunden ist.

```
$ kubectl get pv,pvc
 NAME                                    CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM                           STORAGECLASS         REASON   AGE
 persistentvolume/static-tkg-block-pv    2Gi        RWO            Delete
Bound    default/static-tkg-block-pvc   gc-storage-profile            10s

 NAME                                          STATUS   VOLUME                CAPACITY
ACCESS MODES   STORAGECLASS         AGE
 persistentvolumeclaim/static-tkg-block-pvc   Bound    static-tkg-block-pv   2Gi
RWO            gc-storage-profile   10s
```

Erweiterung persistenter Volumes für TKG-Dienst-Cluster

Sie können die Kubernetes-Volume-Erweiterungsfunktion verwenden, um ein persistentes Block-Volume nach dessen Erstellung zu erweitern. TKG-Dienst-Cluster unterstützen die Erweiterung von Offline- und Online-Volumes.

Info zur Erweiterung persistenter Volumes

Standardmäßig ist `allowVolumeExpansion` bei Speicherklassen, die in der TKG-Clusterumgebung angezeigt werden, auf `true` festgelegt. Dieser Parameter ermöglicht die Konfiguration der Größe eines Online- oder Offline-Volumes.

Ein Volume gilt als offline, wenn es nicht mit einem Knoten oder Pod verbunden ist. Ein Online-Volume ist ein Volume, das auf einem Knoten oder Pod verfügbar ist.

Der Grad der Unterstützung der Volume-Erweiterungsfunktion ist abhängig von der vSphere-Version. Sie können Volumes, die in früheren Versionen von vSphere erstellt wurden, erweitern, wenn Sie ein Upgrade Ihrer vSphere-Umgebung auf entsprechende Versionen durchführen, die Erweiterungen unterstützen.

Hinweis Sie können nur persistente Block-Volumes erweitern. Derzeit unterstützt vSphere IaaS control plane keine Volume-Erweiterung für ReadWriteMany-Volumes.

Beachten Sie beim Erweitern eines persistenten Block-Volumes Folgendes:

- Sie können die Volumes bis zu den durch Speicherkontingente festgelegten Grenzwerten erweitern. vSphere IaaS control plane unterstützt aufeinanderfolgende Skalierungsanforderungen für ein Beanspruchungsobjekt eines persistenten Volumes.
- Alle Arten von Datenspeichern, einschließlich VMFS, vSAN, vSAN Direct, vVols und NFS, unterstützen die Volumeerweiterung.
- Sie können Volumeerweiterungen für Bereitstellungen oder eigenständige Pods durchführen.

- Sie können die Größe von statisch bereitgestellten Volumes in einem Tanzu Kubernetes Grid-Cluster ändern, wenn den Volumes Speicherklassen zugewiesen sind.
- Volumes, die als Teil eines StatefulSet erstellt wurden, können nicht erweitert werden.
- Wenn eine virtuelle Festplatte, die ein Volume-Backing erstellt, über Snapshots verfügt, kann deren Größe nicht geändert werden.
- vSphere IaaS control plane unterstützt keine Volumeerweiterung für strukturbasierte oder migrierte Volumes.

Erweitern eines persistenten Volumes im Online-Modus

Ein Online-Volume ist ein Volume, das auf einem Knoten oder Pod verfügbar ist. Als DevOps Engineer können Sie ein persistentes Onlineblock-Volume erweitern. Tanzu Kubernetes Grid-Cluster unterstützen die Erweiterung von Online-Volumes.

1 Suchen Sie mit dem folgenden Befehl nach der Beanspruchung eines persistenten Volumes, um die Größe zu ändern.

Beachten Sie, dass die Speichergröße, die das Volume in diesem Beispiel verwendet, 1 Gi beträgt.

```
$ kubectl get pv,pvc,pod
NAME                                                        CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS      CLAIM               STORAGECLASS   REASON   AGE
persistentvolume/pvc-5cd51b05-245a-4610-8af4-f07e77fdc984   1Gi        RWO
Delete           Bound       default/block-pvc   block-sc                4m56s

NAME                              STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/block-pvc   Bound    pvc-5cd51b05-245a-4610-8af4-f07e77fdc984
1Gi        RWO            block-sc       5m3s

NAME            READY   STATUS    RESTARTS   AGE
pod/block-pod   1/1     Running   0          26s
```

2 Patchen Sie das PVC, um es zu vergrößern. Erhöhen Sie den Wert für die Größe beispielsweise auf 2 Gi.

Diese Aktion löst eine Erweiterung in dem Volume aus, das mit dem PVC verknüpft ist.

```
$ kubectl patch pvc block-pvc -p '{"spec": {"resources": {"requests": {"storage":
"2Gi"}}}}'
persistentvolumeclaim/block-pvc edited
```

3 Stellen Sie sicher, dass sowohl die PVC- als auch die PV-Größe erhöht wurde.

```
$ kubectl get pvc,pv,pod
NAME                              STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/block-pvc   Bound    pvc-5cd51b05-245a-4610-8af4-f07e77fdc984
2Gi        RWO            block-sc       6m18s
```

```
NAME                                                        CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS      CLAIM               STORAGECLASS   REASON   AGE
persistentvolume/pvc-5cd51b05-245a-4610-8af4-f07e77fdc984   2Gi        RWO
Delete           Bound       default/block-pvc   block-sc                6m11s

NAME            READY   STATUS    RESTARTS   AGE
pod/block-pod   1/1     Running   0          101s
```

4 Verwenden Sie die vSphere Client, um die Größe des neuen persistenten Volumes zu überprüfen.

Weitere Informationen finden Sie unter Überwachen der Volume-Integrität in einem Tanzu Kubernetes Grid-Cluster.

Erweitern eines persistenten Volumes im Offline-Modus

Ein Volume gilt als offline, wenn es nicht mit einem Knoten oder Pod verbunden ist. Tanzu Kubernetes Grid-Cluster unterstützen die Erweiterung von Offline-Volumes.

1 Erstellen Sie eine Beanspruchung eines persistenten Volumes (Persistent Volume Claim, PVC) mit einer bestehenden Speicherklasse.

Im Beispiel beträgt die Größe des angeforderten Speichers 1 Gi.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-block-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: example-block-sc
```

```
kubectl apply -f example-block-pvc.yaml
```

2 Patchen Sie das PVC, um es zu vergrößern.

Wenn das MATERIAL nicht an einen Knoten angehängt ist oder von einem Pod verwendet wird, verwenden Sie den folgenden Befehl zum Patchen von PVC. In diesem Beispiel beträgt die angeforderte Speichererhöhung 2 Gi.

Diese Aktion löst eine Erweiterung in dem Volume aus, das mit dem PVC verknüpft ist.

```
kubectl patch pvc example-block-pvc -p '{"spec": {"resources": {"requests": {"storage":
"2Gi"}}}}'
```

3 Vergewissern Sie sich, dass das Volume größer geworden ist.

```
kubectl get pv
NAME                                       CAPACITY ACCESS MODES RECLAIM POLICY STATUS
CLAIM                         STORAGECLASS           REASON AGE
pvc-9e9a325d-ee1c-11e9-a223-005056ad1fc1   2Gi            RWO         Delete     Bound
default/example-block-pvc   example-block-sc              6m44s
```

4 Stellen Sie sicher, dass die Größenänderung des PVC aussteht.

Hinweis Die Größe des PVC bleibt unverändert, bis das PVC von einem Pod verwendet wird.

Das folgende Beispiel zeigt, dass sich die PVC-Größe nicht geändert hat, da sie nicht von einem Pod verwendet wurde. Wenn Sie `kubectl describe pvc` ausführen, wird die `FilesystemResizePending`-Bedingung auf das PVC angewendet. Sobald sie von einem Pod verwendet wird, ändert sich die Größe.

```
kubectl get pvc
NAME                STATUS VOLUME                                     CAPACITY ACCESS
MODES   STORAGECLASS       AGE
example-block-pvc   Bound  pvc-9e9a325d-ee1c-11e9-a223-005056ad1fc1   1Gi
RWO       example-block-sc   6m57s
```

5 Erstellen Sie einen Pod zur Verwendung des PVC.

Wenn das PVC vom Pod verwendet wird, wird das Dateisystem erweitert.

6 Vergewissern Sie sich, dass die PVC-Größe geändert wurde.

```
kubectl get pvc
NAME                STATUS VOLUME                                     CAPACITY ACCESS MODES
STORAGECLASS     AGE
example-block-pvc   Bound  pvc-24114458-9753-428e-9c90-9f568cb25788   2Gi         RWO
example-block-sc 2m12s
```

Die Bedingung `FilesystemResizePending` wurde aus dem PVC entfernt. Volume-Erweiterung ist abgeschlossen.

7 Verwenden Sie die vSphere Client, um die Größe des neuen persistenten Volumes zu überprüfen.

Weitere Informationen finden Sie unter Überwachen der Volume-Integrität in einem Tanzu Kubernetes Grid-Cluster.

Verwalten von Netzwerken für TKG-Dienstcluster

17

Dieser Abschnitt enthält Informationen zum Verwalten von Netzwerken für TKG-Dienstcluster.

Lesen Sie als Nächstes die folgenden Themen:

- Installieren des NSX-Verwaltungs-Proxy-Diensts
- Aktivieren des Antrea-NSX-Adapters für einen TKG-Dienstcluster
- Festlegen der Standard-CNI für Tanzu Kubernetes-Cluster
- Anpassen der TKG-Dienstkonfiguration für TKG-Cluster
- NSX-Netzwerkobjekte für TKG-Cluster

Installieren des NSX-Verwaltungs-Proxy-Diensts

Sie können den NSX-Verwaltungs-Proxy zusammen mit dem Antrea-NSX-Adapter verwenden, um NSX Manager über Antrea-basierte TKG-Dienstcluster zu erreichen. Der NSX-Verwaltungs-Proxy ist erforderlich, wenn eine Isolierung zwischen den Supervisor-Verwaltungs- und Arbeitslastnetzwerken besteht. Der Proxy gilt für alle TKGS-Cluster, die nach der Bereitstellung des NSX-Verwaltungs-Proxy-Diensts erstellt wurden.

Anwendungsfall

Der NSX-Verwaltungs-Proxy ist für die Verwendung in Verbindung mit dem Aktivieren des Antrea-NSX-Adapters für einen TKG-Dienstcluster vorgesehen.

Wenn Supervisor wie in der Regel getrennt zwischen den Verwaltungs- und Arbeitslastnetzwerken bereitgestellt wird, um die NSX-Verwaltungsebene von einem mit dem Antrea-NSX-Adapter konfigurierten TKGS-Cluster zu erreichen, müssen Sie den NSX-Verwaltungs-Proxy verwenden. Wenn das System diesen Proxy erkennt, übergibt Supervisor den Proxy automatisch an die Antrea-NSX-Adapterkonfiguration. Wenn der Proxy nicht installiert ist, kann der Antrea-NSX-Adapter nicht gestartet werden, wenn eine Verwaltungsnetzwerkisolierung vorliegt.

Sobald der NSX-Verwaltungs-Proxy-Dienst bereitgestellt wurde, können Sie den Antrea-NSX-Adapter installieren. Weitere Informationen hierzu finden Sie unter Aktivieren des Antrea-NSX-Adapters für einen TKG-Dienstcluster.

Voraussetzungen

Der NSX-Verwaltungs-Proxy wird als Supervisor-Dienst installiert. Zum Installieren des Proxy-Diensts müssen die folgenden Voraussetzungen erfüllt sein:

- vSphere 8 U3 (8.0.3) oder höher
- NSX 4.1 oder höher
- Supervisor ist mit NSX-Netzwerk aktiviert
- Berechtigung „Supervisor-Dienste verwalten“ auf vCenter Server
- Kenntnisse der Supervisor-Dienste

Herunterladen der erforderlichen YAML-Dateien

Laden Sie die erforderlichen YAML-Dateien herunter, einschließlich der Dienstdefinition und der Datenwerte.

1. Navigieren Sie zur Supervisor-Dienste-Verteilungs-Site unter https://www.vmware.com/go/supervisor-service.
2. Führen Sie einen Bildlauf zum Abschnitt NSX-Verwaltungs-Proxy durch und laden Sie die folgenden Dateien herunter:
   a. Die Definitionsdatei des NSX-Verwaltungs-Proxy-Diensts: `nsx-management-proxy.yml`
   b. Die Konfigurationsdatei des NSX-Verwaltungs-Proxy-Diensts: `nsx-management-proxy-data-values.yml`

Registrieren des NSX-Verwaltungs-Proxys als Dienst

Führen Sie diese Schritte aus, um den NSX-Verwaltungs-Proxy als Supervisor-Dienst zu registrieren.

1. Navigieren Sie mit dem vSphere Client zu **Arbeitslastverwaltung** > **Dienste**.
2. Wählen Sie **Neuen Dienst hinzufügen** > **Hinzufügen** aus.
3. Klicken Sie auf **Hochladen**.
4. Navigieren Sie zur heruntergeladenen Datei `nsx-management-proxy.yml`, und wählen Sie sie aus.
5. Stellen Sie sicher, dass die Definition des NSX-Verwaltungs-Proxy-Diensts erfolgreich hochgeladen wurde.
6. Klicken Sie auf **Beenden**.
7. Stellen Sie sicher, dass die Registrierungskarte für den NSX-Verwaltungs-Proxy-Dienst auf der Registerkarte **Dienste** aufgeführt wird.

Konfigurieren des NSX-Verwaltungs-Proxy-Diensts

Aktualisieren Sie vor der Installation des NSX Manager-Proxy-Diensts die zugehörige Datenwertdatei mit den für Ihre Umgebung geeigneten Konfigurationswerten.

1 Öffnen Sie mithilfe eines Texteditors die Datei `nsx-management-proxy-data-values.yml`.

2 Bearbeiten Sie die Eigenschaften entsprechend der in der folgenden Tabelle beschriebenen Umgebung.

3 Speichern Sie die Änderungen.

Name	Wert
nsxManagers	Liste der NSX Manager-IP-Adressen (erforderlich). Sie müssen eine tatsächliche IP-Adresse und nicht die virtuelle IP-Adresse (VIP) verwenden. Wenn Sie einen NSX-Verwaltungscluster verwenden, müssen Sie alle drei tatsächlichen IP-Adressen in die Liste aufnehmen.
loadBalancerIP	IP aus dem IP-Pool des Supervisor-Lastausgleichsdiensts (optional). Die IP-Adresse für dieses Feld wird vom „Ingress“-CIDR für das Arbeitslastnetzwerk getrennt. Wenn ein Arbeitslastnetzwerk anfänglich während der Supervisor-Aktivierung oder später beim Erstellen eines vSphere-Namespace und Überschreiben der Netzwerkeinstellungen erstellt wird, gibt es eine Konfigurationseinstellung für „Ingress“. Diese akzeptiert einen IP-CIDR-Block, der zum Zuteilen von IP-Adressen für Dienste verwendet wird. Diese wiederum werden über den Lastausgleichsdienst-Typ und Ingress über alle die in dieser Supervisor-Instanz erstellten vSphere Namespaces hinweg veröffentlicht. Wenn das Feld „loadBalancerIP“ nicht angegeben ist, teilt das System automatisch eine verfügbare IP-Adresse aus dem CIDR-Bereich „Ingress“ zu. Wenn „loadBalancerIP“ angegeben wird, muss sie innerhalb des CIDR-Bereichs „Ingress“ liegen und darf nicht mit bereits zugeteilten IP-Adressen in Konflikt stehen. Sie können zugeteilte „Ingress“-IPs mithilfe des Befehls „kubectl get services -o wide -A“ im vSphere-Namespace anzeigen. Die IPs befinden sich in der Spalte „EXTERNAL-IP“ der kubectl-Ausgabe.

Installieren des NSX-Verwaltungs-Proxy-Diensts

Installieren Sie den NSX-Verwaltungs-Proxy-Dienst, indem Sie die folgenden Schritte ausführen.

1 Navigieren Sie zum Bildschirm **Arbeitslastverwaltung > > Dienste**.

2 Wählen Sie auf der Dienstkarte **NSX-Verwaltungs-Proxy** die Option **Aktionen > Auf Supervisor installieren** aus.

3 Wählen Sie die Registerkarte **Verfügbar** aus.

4 Kopieren Sie den bearbeiteten Inhalt aus der Datei `nsx-management-proxy-data-values.yml` und fügen Sie ihn im Eingabefeld „YAML-Dienstkonfiguration“ ein.

5 Klicken Sie auf **OK**, um mit der Installation von Harbor fortzufahren.

6 Überwachen und überprüfen Sie die Installation.

   Überwachen Sie die Installation, indem Sie das Feld Supervisoren auf der Karte des NSX-Verwaltungs-Proxy-Diensts überprüfen. Die Zahl sollte neben Supervisoren-Schritten angezeigt werden. Der Dienst befindet sich im Konfigurationszustand, bis der gewünschte Zustand erreicht ist. Wenn der gewünschte Zustand erreicht ist, ändert sich der Status des Diensts in „Konfiguriert“.

7 Stellen Sie sicher, dass ein vSphere-Namespace für den NSX-Verwaltungs-Proxy vorhanden ist.

   Nach der Installation des NSX-Verwaltungs-Proxys wird ein vSphere-Namespace für die Dienstinstanz erstellt.

8 Rufen Sie die IP-Adresse des Proxy-Lastausgleichsdiensts ab.

   Sie können die IP-Adresse des Proxy-Lastausgleichsdiensts auf der Registerkarte **Netzwerk** des vSphere-Namespace des NSX-Verwaltungs-Proxys anzeigen.

Fehlerbehebung beim NSX-Verwaltungs-Proxy-Dienst

Folgende Fehlermeldung weist darauf hin, dass die Umgebung nicht kompatibel ist. Stellen Sie sicher, dass die Umgebung mit den im Abschnitt „Voraussetzungen“ aufgeführten Versionen übereinstimmt.

```
Creation of Supervisor Service with ID nsx-management-proxy.nsx.vmware.com is not allowed.
Only service IDs defined in the allow-list file /etc/vmware/wcp/supervisor-services-allow-
list.txt are allowed.
```

Aktivieren des Antrea-NSX-Adapters für einen TKG-Dienstcluster

In diesem Thema finden Sie Informationen zum Aktivieren des Antrea-NSX-Adapters, mit dem Sie einen TKG-Dienstcluster integrieren können, der die Antrea-CNI mit NSX Manager für die Netzwerkverwaltung und -überwachung verwendet.

Voraussetzungen für den Antrea-NSX-Adapter

Beachten Sie die folgenden Voraussetzungen:

- vSphere 8 U3 (8.0.3) oder höher
- NSX 4.1 oder höher
- Supervisor ist mit NSX-Netzwerk aktiviert

- TKG-Dienst 3.0 oder höher
- Tanzu Kubernetes-Version v1.28.x für vSphere 8.x oder höher
- Installieren des NSX-Verwaltungs-Proxy-Diensts, wenn zwischen Verwaltungs- und Arbeitslastnetzwerken eine Isolierung besteht, was die typische Supervisor-Topologie darstellt

Anforderungen für den Antrea-NSX-Adapter

Mit dem Antrea-NSX-Adapter können Sie einen Antrea-basierten TKG-Dienstcluster in NSX Manager integrieren. Nach der Konfiguration des Adapters können Sie das Netzwerkverhalten des Clusters mithilfe von NSX Manager verwalten. Weitere Informationen zu den Funktionen der Antrea-NSX-Integration finden Sie unter Integration von Kubernetes-Clustern in die Antrea-CNI im Administratorhandbuch für NSX 4.1.

Sie müssen den Antrea-NSX-Adapter für jeden TKG-Dienstcluster aktivieren, den Sie in NSX integrieren möchten. Anders ausgedrückt, besteht zwischen einem Adapter und einem Cluster ein 1-zu-1-Verhältnis. Darüber hinaus können Sie den Antrea-NSX-Adapter nur mit neuen Clusterbereitstellungen verwenden. Sie müssen den Adapter vor der Erstellung des TKG-Dienstclusters aktivieren. Darüber hinaus müssen Sie den Namen des zu erstellenden Clusters in die Definition der Adapterressource aufnehmen.

Wenn die NSX Tier-0- oder Tier-1-Gateways für TKGS-Cluster mit einer SNAT-IP konfiguriert sind, teilen sich alle Antrea-NSX-Verbindungen eine einzige Quell-IP. NSX interpretiert dies als eine Vielzahl von Control Plane-Verbindungen von derselben IP und trennt die Verbindungen. Wenn dies der Fall ist, müssen Sie die NSX UA-Firewallregeln manuell anpassen. Weitere Informationen finden Sie im folgenden KB-Artikel: https://knowledge.broadcom.com/external/article?articleNumber=317179.

Aktivieren des Antrea-NSX-Adapters

Führen Sie die folgenden Schritte durch, um den Antrea-NSX-Adapter zu aktivieren.

1 Authentifizieren Sie sich mithilfe von kubectl bei Supervisor.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-username USERNAME
```

2 Wechseln Sie den Kontext zu dem Ziel-vSphere-Namespace, in dem Sie den TKG-Dienstcluster bereitstellen möchten.

```
kubectl config use-context vsphere-namespace
```

3 Erstellen Sie die benutzerdefinierte Ressourcendefinition `AntreaConfig.yaml`.

```
#AntreaConfig.yaml
apiVersion: cni.tanzu.vmware.com/v1alpha1
kind: AntreaConfig
metadata:
 name: tkgs-cluster-name-antrea-package #prefix required
```

```
 namespace: tkgs-cluster-ns
spec:
  antreaNSX:
    enable: true #false by default
```

Dabei gilt:

- Das Element `antreaNSX.enable.true` aktiviert den Antrea-NSX-Adapter. Dieses Feld ist standardmäßig auf „False“ festgelegt.
- Das Element `metadata.name` enthält das obligatorische Suffix `-antrea-package` und als Präfix den genauen Namen des TKG-Dienstclusters, der von Ihnen erstellt und in dem der Adapter verwendet wird.

4 Wenden Sie die benutzerdefinierte Ressourcendefinition „AntreaConfig“ an.

```
kubectl apply -f AntreaConfig.yaml
```

5 Stellen Sie den TKG-Dienstcluster bereit.

Der Adapter kann mit der v1alpha3-API oder der v1beta1-API verwendet werden.

Weitere Informationen hierzu finden Sie unter Workflow zum Bereitstellen von TKG-Clustern auf mithilfe von Kubectl.

6 Stellen Sie die Funktionsfähigkeit des Adapters sicher, indem Sie sich bei NSX Manager anmelden.

Weitere Informationen finden Sie in der Dokumentation zu NSX 4.1.

Festlegen der Standard-CNI für Tanzu Kubernetes-Cluster

Die standardmäßige Container-Netzwerkschnittstelle (Container Network Interface, CNI) für Tanzu Kubernetes-Cluster ist Antrea. Mithilfe des vSphere Client können Sie die Standard-CNI ändern.

Standard-CNI

vSphere IaaS control plane unterstützt zwei CNI-Optionen für TKG Cluster: Antrea und Calico. Die vom System festgelegte Standard-CNI ist Antrea.

Sie können die Standard-CNI mit dem vSphere Client ändern. Gehen Sie wie folgt vor, um die Standard-CNI festzulegen.

Vorsicht Das Ändern der Standard-CNI ist ein globaler Vorgang. Der neu festgelegte Standard gilt für alle neuen TKG-Cluster, die vom Dienst erstellt wurden. Vorhandene Cluster bleiben unverändert.

1 Melden Sie sich über vSphere IaaS control plane bei der vSphere Client-Umgebung an.

2 Wählen Sie **Arbeitslastverwaltung** und anschließend **Supervisoren** aus.

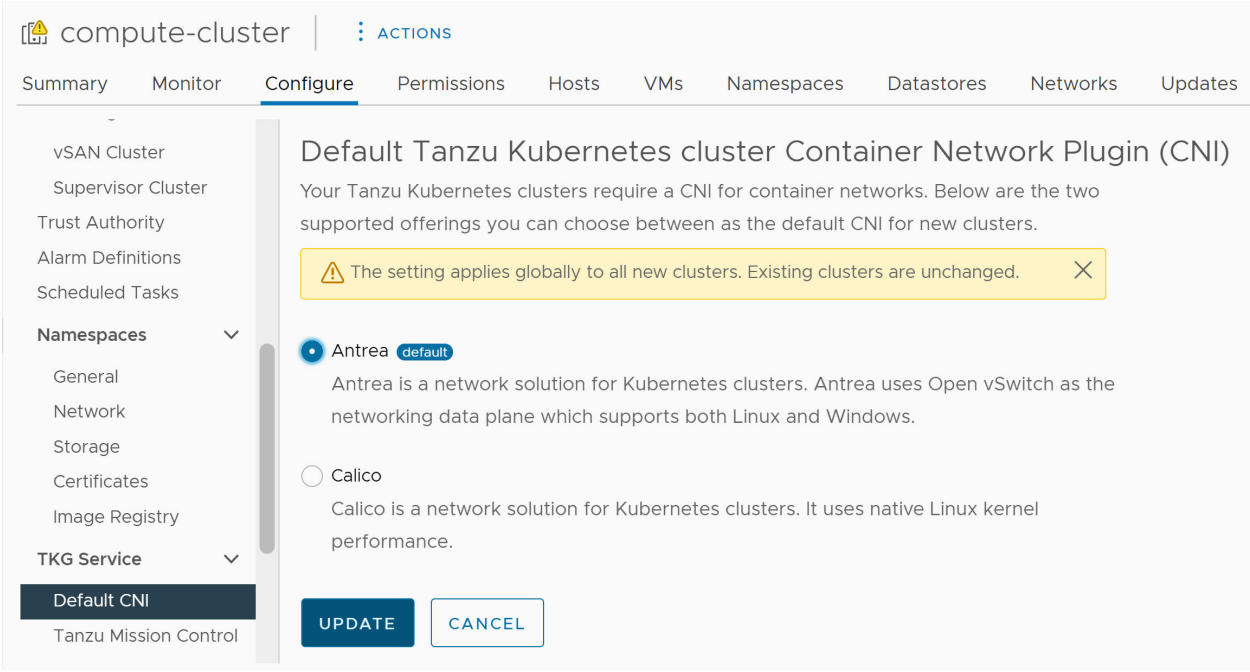
3 Wählen Sie die Supervisor-Instanz aus der Liste aus.

4 Wählen Sie **Konfigurieren** und anschließend **TKG-Dienst > Standard-CNI** aus.

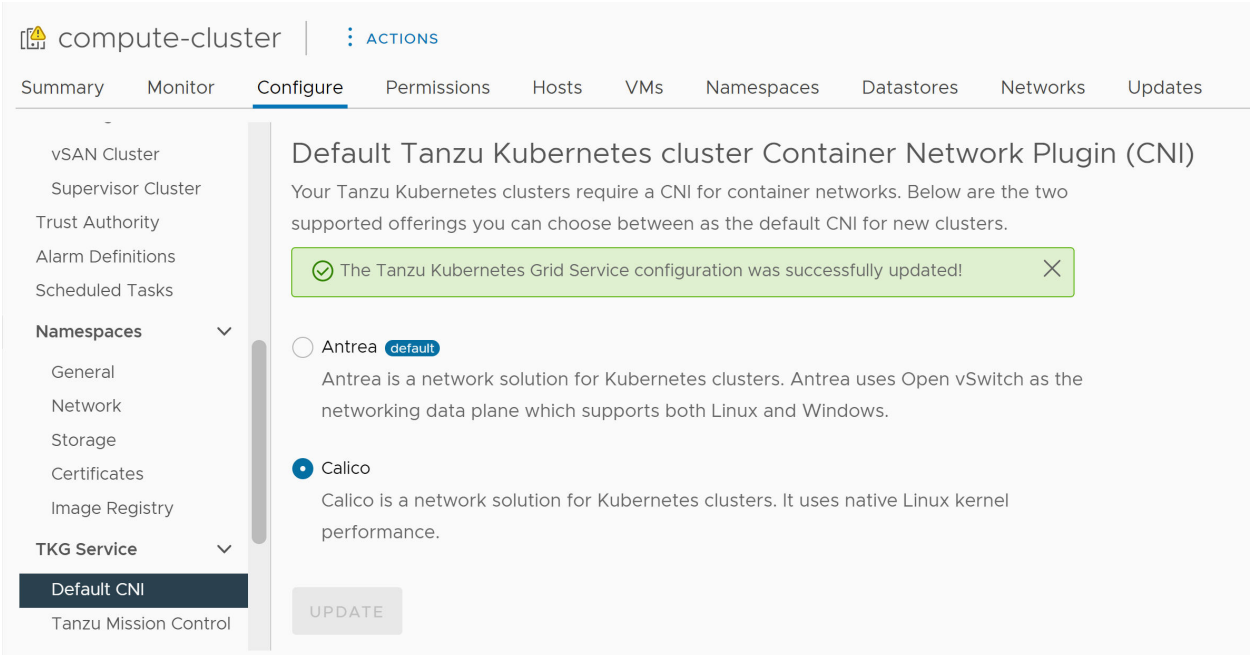
5 Wählen Sie die Standard-CNI für neue Cluster aus.

6 Klicken Sie auf **Aktualisieren**.

Die folgende Abbildung zeigt die Auswahl der Standard-CNI.



Die folgende Abbildung zeigt die Änderung der CNI-Auswahl von Antrea zu Calico.



Anpassen der TKG-Dienstkonfiguration für TKG-Cluster

Sie können die TKG-Dienstkonfiguration für TKG-Cluster anpassen, die mithilfe der v1alpha3-API bereitgestellt werden, einschließlich Containernetzwerkschnittstelle (CNI), Proxyserver und TLS-Zertifikaten.

Informationen zum Anpassen der TkgServiceConfiguration

Sie können globale Einstellungen für Tanzu Kubernetes-Cluster konfigurieren, indem Sie die TkgServiceConfiguration bearbeiten. Mit dieser Konfiguration können Sie die Standard-CNI festlegen und einen globalen Proxy-Server hinzufügen und ein oder mehrere vertrauenswürdige TLS-Zertifikate hinzufügen.

Vorsicht Das Anpassen der TkgServiceConfiguration ist ein globaler Vorgang. Alle Änderungen, die Sie am `TkgServiceConfiguration`-Objekt vornehmen, gelten für alle TKG-Cluster, die von diesem Dienst bereitgestellt werden. Wenn ein rollierendes Update entweder manuell oder durch ein Upgrade initiiert wird, werden Cluster durch die geänderte Dienstspezifikation aktualisiert.

Hinweis Das Anpassen der TkgServiceConfiguration gilt für Tanzu Kubernetes-Cluster, die mit der v1alpha3-API bereitgestellt werden. Es gilt nicht für mit der v1beta1-API bereitgestellte Cluster.

TkgServiceConfiguration-Spezifikation

Die `TkgServiceConfiguration`-Spezifikation enthält Felder für die Konfiguration der Tanzu Kubernetes Grid-Instanz.

Wichtig Ein gültiger Schlüsselname darf nur aus alphanumerischen Zeichen, einem Bindestrich (z. B. `key-name`), einem Unterstrich (z. B. `KEY_NAME`) oder einem Punkt (z. B. `key.name`) bestehen. Leerzeichen können in Schlüsselnamen nicht verwendet werden.

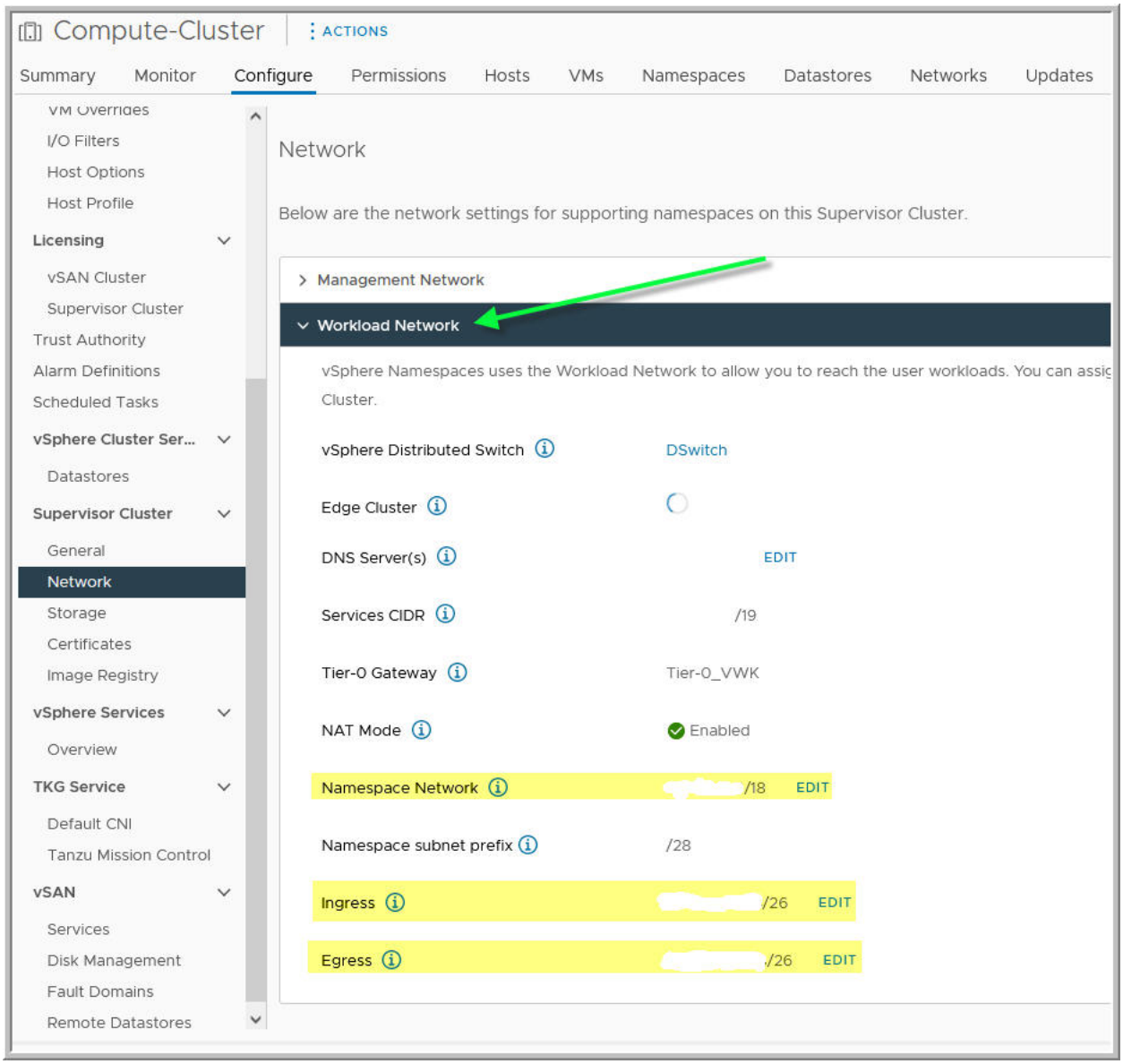
```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TkgServiceConfiguration
metadata:
  name: tkg-service-configuration-spec
spec:
  defaultCNI: string
  proxy:
    httpProxy: string
    httpsProxy: string
    noProxy: [string]
  trust:
    additionalTrustedCAs:
      - name: string
        data: string
  defaultNodeDrainTimeout: time
```

TkgServiceConfiguration-Spezifikation mit Anmerkungen

In der folgenden YAML werden die konfigurierbaren Felder für jeden der `TkgServiceConfiguration`-Spezifikationsparameter aufgelistet und beschrieben.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TkgServiceConfiguration
#valid config key must consist of alphanumeric characters, '-', '_' or '.'
metadata:
  name: tkg-service-configuration-spec
spec:
  #defaultCNI is the default CNI for all Tanzu Kubernetes
  #clusters to use unless overridden on a per-cluster basis
  #supported values are antrea, calico, antrea-nsx-routed
  #defaults to antrea
  defaultCNI: string
  #proxy configures a proxy server to be used inside all
  #clusters provisioned by this TKGS instance
  #if implemented all fields are required
  #if omitted no proxy is configured
  proxy:
    #httpProxy is the proxy URI for HTTP connections
    #to endpionts outside the clusters
    #takes the form http://<user>:<pwd>@<ip>:<port>
    httpProxy: string
    #httpsProxy is the proxy URI for HTTPS connections
    #to endpoints outside the clusters
    #takes the frorm http://<user>:<pwd>@<ip>:<port>
    httpsProxy: string
    #noProxy is the list of destination domain names, domains,
    #IP addresses, and other network CIDRs to exclude from proxying
    #must include from Workload Network: [Namespace Netowrk, Ingress, Egress]
    noProxy: [string]
  #trust configures additional trusted certificates
  #for the clusters provisioned by this TKGS instance
  #if omitted no additional certificate is configured
  trust:
    #additionalTrustedCAs are additional trusted certificates
    #can be additional CAs or end certificates
    additionalTrustedCAs:
      #name is the name of the additional trusted certificate
      #must match the name used in the filename
      - name: string
        #data holds the contents of the additional trusted cert
        #PEM Public Certificate data encoded as a base64 string
        data: string
  #defaultNodeDrainTimeout is the total amount of time the
  #controller spends draining a node; default is undefined
  #which is the value of 0, meaning the node is drained
  #without any time limitations; note that `nodeDrainTimeout`
  #is different from `kubectl drain --timeout`
  defaultNodeDrainTimeout: time
```

Abbildung 17-1. Erforderliche NoProxy-Werte



Hinweis Wenn ein globaler Proxy für die `TkgServiceConfiguration` konfiguriert ist, werden diese Proxy-Informationen nach der ersten Bereitstellung des Clusters an das Clustermanifest weitergegeben. Die globale Proxy-Konfiguration wird nur dann zum Clustermanifest hinzugefügt, wenn beim Erstellen des Clusters keine Proxy-Konfigurationsfelder vorhanden sind. Anders ausgedrückt: Die Konfiguration pro Cluster hat Vorrang und überschreibt eine globale Proxy-Konfiguration.

Beispiel für eine TkgServiceConfiguration-Spezifikation

In der folgenden YAML werden die konfigurierbaren Felder für jeden der `TkgServiceConfiguration`-Spezifikationsparameter aufgelistet und beschrieben.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TkgServiceConfiguration
metadata:
  name: tkgserviceconfiguration_example
spec:
  defaultCNI: calico
  proxy:
    #supported format is `http://<user>:<pwd>@<ip>:<port>`
    httpProxy: http://admin:PaSsWoRd@10.66.100.22:80
    httpsProxy: http://admin:PaSsWoRd@10.66.100.22:80
    #noProxy vaulues are from Workload Network: [Namespace Network, Ingress, Egress]
    noProxy: [10.246.0.0/16,192.168.144.0/20,192.168.128.0/20]
  trust:
    additionalTrustedCAs:
      #name is the name of the public cert
      - name: CompanyInternalCA-1
      #data is base64-encoded string of a PEM encoded public cert
        data: LS0tLS1C...LS0tCg==
        #where "..." is the middle section of the long base64 string
      - name: CompanyInternalCA-2
        data: MTLtMT1C...MT0tPg==
  defaultNodeDrainTimeout: 0
```

Bearbeiten der TkgServiceConfiguration

Befolgen Sie das folgende Verfahren, um die TkgServiceConfiguration-Spezifikation zu bearbeiten.

1 Konfigurieren Sie die Kubectl-Bearbeitung. Weitere Informationen hierzu finden Sie unter Konfigurieren eines Texteditors für Kubectl.

2 Authentifizieren Sie sich beim Supervisor.

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

3 Führen Sie einen Kontextwechsel zum Ziel-vSphere-Namespace durch.

```
kubectl config use-context vSphere-Namespace
```

4 Rufen Sie die TkgServiceConfiguration-Spezifikation ab.

```
kubectl get tkgserviceconfigurations
```

5 Starten Sie den Ladevorgang, um die TkgServiceConfiguration-Spezifikation zu bearbeiten.

```
kubectl edit tkgserviceconfigurations tkg-service-configuration
```

Das System öffnet die `tkg-service-configuration`-Spezifikation im Standardtexteditor, der durch die Umgebungsvariable KUBE_EDITOR oder EDITOR definiert wird.

6 Bearbeiten Sie die TkgServiceConfiguration entsprechend Ihren Anforderungen.

7 Um die Änderungen anzuwenden, speichern Sie die Datei im Texteditor. Schließen Sie zum Abbrechen den Editor ohne Speichern.

Wenn Sie die Änderung im Texteditor speichern, aktualisiert kubectl die `tkg-service-configuration`-Dienstspezifikation.

8 Stellen Sie sicher, dass die TkgServiceConfiguration-Spezifikation aktualisiert wird.

```
kubectl get tkgserviceconfigurations
```

Propagieren von globalen Konfigurationsänderungen an vorhandenen Clustern

Wenn in der `TkgServiceConfiguration` globale Einstellungen vorgenommen werden, werden diese möglicherweise nicht automatisch an die vorhandenen Cluster weitergegeben.
Wenn Sie beispielsweise Änderungen an den `proxy`- oder `trust`-Einstellungen in der `TkgServiceConfiguration` vornehmen, wirken sich diese Änderungen möglicherweise nicht auf bereits bereitgestellte Cluster aus.

Um eine globale Änderung manuell an einen vorhandenen Cluster weiterzugeben, müssen Sie den Tanzu Kubernetes-Cluster patchen, damit der Cluster die an der `TkgServiceConfiguration` vorgenommenen Änderungen übernimmt.

Beispiel:

```
kubectl patch tkc <CLUSTER_NAME> -n <NAMESPACE> --type merge -p "{\"spec\":{\"settings\":
{\"network\":{\"proxy\": null}}}}"
```

```
kubectl patch tkc <CLUSTER_NAME> -n <NAMESPACE> --type merge -p "{\"spec\":{\"settings\":
{\"network\":{\"trust\": null}}}}"
```

NSX-Netzwerkobjekte für TKG-Cluster

Dieses Thema enthält eine Liste der Netzwerkobjekte, die für einen TKG-Cluster erstellt werden, wenn Sie den Supervisor mit einem NSX-Netzwerk verwenden.

NSX-Netzwerkobjekte für TKG-Cluster

Jeder TKG-Cluster sollte über die folgenden Netzwerkressourcen verfügen: ein virtuelles Netzwerk, eine virtuelle Netzwerkschnittstelle und einen VM-Dienst.

Das System stellt automatisch einen eingebetteten NSX-Lastausgleichsdienst bereit, wenn vSphere IaaS control plane aktiviert ist und eine Supervisor-Instanz bereitgestellt wird. Dieser Lastausgleichsdienst ist für die Supervisor-Steuerungsebene vorgesehen und bietet Zugriff auf den Kubernetes-API-Server.

Wenn Sie einen Kubernetes-Dienst vom Typ „LoadBalancer“ für einen TKG-Cluster erstellen, wird ein eingebetteter NSX-Lastausgleichsdienst für diesen Dienst bereitgestellt.

Netzwerkobjekt	Netzwerkressourcen	Beschreibung
VirtualNetwork	Tier-1-Router und verknüpftes Segment	Knotennetzwerk für den Cluster
VirtualNetworkInterface	Logischer Port auf Segment	Knotennetzwerkschnittstelle für Clusterknoten
VirtualMachineService	Nicht verfügbar	VirtualMachineService wird erstellt und in einen k8s-Dienst übersetzt.
Dienst	Lastausgleichsserver mit VirtualServer-Instanz und zugeordneter Serverpool (Mitgliederpool)	Der Kubebernetes-Dienst vom Typ „Lastausgleichsdienst“ wird für den Zugriff auf den TKG-Cluster-API-Server erstellt.
Endpoints	Die Endpoint-Mitglieder (Knoten der Steuerungsebene des TKG-Clusters) sollten sich im Mitgliedspool befinden.	Ein Endpoint wird erstellt, um alle Steuerungsebenenknoten des TKG-Clusters einzubeziehen.
VirtualMachineService in Supervisor	Nicht verfügbar	Ein VirtualMachineService wird im Supervisor erstellt und in einen Kubernetes-Dienst im Supervisor übersetzt
Lastausgleichsdienst im Supervisor	VirtualServer im TKG-Cluster-Lastausgleichsdienst und ein zugehöriger Mitgliederpool.	Der Lastausgleichsdienst wird im Supervisor für den Zugriff auf diesen LB-Diensttyp erstellt
Endpoints im Supervisor	Die Endpoint-Mitglieder (TKG-Cluster-Worker-Knoten) sollten sich im Mitgliedspool in NSX befinden.	Ein Endpoint wird erstellt, um alle Worker-Knoten des TKG-Clusters einzubeziehen
Lastausgleichsdienst im TKG-Cluster	Nicht verfügbar	Für den Lastausgleichsdienst im vom Benutzer bereitgestellten TKG-Cluster sollte der Status mit der Lastausgleichsdienst-IP aktualisiert werden

Knotennetzwerk

Für jeden TKG-Cluster müssen die folgenden Netzwerkobjekte und zugeordneten NSX-Ressourcen erstellt worden sein.

Netzwerkobjekt	NSX-Ressourcen	Beschreibung	IPAM
VirtualNetwork	Tier-1-Gateway und verknüpftes Segment	Knotennetzwerk für den TKG-Cluster	SNAT-IP ist zugewiesen
VirtualNetworkInterface	Logischer Port auf dem verknüpften Segment	Knotennetzwerkschnittstelle für TKG-Clusterknoten	Jedem Knoten wird eine IP zugewiesen

Lastausgleichsdienst der Steuerungsebene

Netzwerkobjekt	Netzwerkressourcen	Beschreibung	IPAM
VirtualMachineService	Nicht verfügbar	VirtualMachineService wird erstellt und in einen Kubernetes-Dienst übersetzt.	Enthält die Lastausgleichsdienst-VIP
Dienst	Lastausgleichsserver mit VirtualServer-Instanz und zugeordneter Serverpool (Mitgliederpool)	Der Kubebernetes-Dienst vom Typ „Lastausgleichsdienst“ wird für den Zugriff auf den TKG-Cluster-API-Server erstellt.	Externe IP ist zugewiesen.
Endpoints	Die Endpoint-Mitglieder sind die Knoten der Steuerungsebene des TKG-Clusters und sollten sich im Mitgliedspool befinden.	Ein Endpoint wird erstellt, um alle Steuerungsebenenknoten des TKG-Clusters einzubeziehen.	Nicht verfügbar

NSX Load Balancer

Für jeden erstellten TKG-Cluster erstellt das System eine einzelne Instanz eines kleinen NSX Load Balancers. Dieser Lastausgleichsdienst enthält die in der folgenden Tabelle aufgeführten Objekte:

Objektnummer	Beschreibung
1	Virtueller Server (VS) für den Zugriff auf die Kubernetes-Steuerungsebenen-API auf Port 8443.
1	Serverpool mit den 3 Kubernetes-Steuerungsebenenknoten.
1	VS für HTTP-Ingress-Controller.
1	VS für HTTPS-Ingress-Controller.

NAT-Regeln

Für jeden erstellten TKG-Cluster definiert das System die folgenden NSX-NAT-Regeln auf dem logischen Tier-0-Router:

Objektnummer	Beschreibung
1	Für jeden Kubernetes-Namespace erstellte SNAT-Regel, die 1 IP aus dem dynamischen IP-Pool als übersetzte IP-Adresse verwendet.
1	(Nur NAT-Topologie) Für jeden Kubernetes-Cluster erstellte SNAT-Regel, die 1 IP aus dem dynamischen IP-Pool als übersetzte IP-Adresse verwendet. Das Kubernetes-Cluster-Subnetz wird mithilfe einer /24-Netzmaske aus dem IP-Block der Knoten abgeleitet.

DFW-Regeln

Für jeden erstellten TKG-Cluster definiert das System die folgenden NSX Distributed Firewall-Regeln:

Objektnummer	Beschreibung
1	DFW-Regel für `kube-dns`, angewendet auf den logischen CoreDNS-Pod-Port:
1	DFW-Regel für den Validator im Namespace, angewendet auf den logischen Validator-Pod-Port:

Verwalten von Sicherheit für TKG-Dienstcluster

18

Dieser Abschnitt enthält Informationen zum Verwalten von Sicherheit für TKG-Dienstcluster.

Lesen Sie als Nächstes die folgenden Themen:

- Sicherheit für TKG-Dienstcluster
- Konfigurieren von PSA für TKR 1.25 und höher
- Konfigurieren von PSP für TKR 1.24 und früher
- Anwenden der standardmäßigen Pod-Sicherheitsrichtlinie auf TKG-Dienstcluster
- Verwalten von TLS-Zertifikaten für TKG-Dienstcluster
- Rotieren von NSX-Zertifikaten

Sicherheit für TKG-Dienstcluster

Der TKG-Dienst auf Supervisor nutzt vSphere-Sicherheitsfunktionen und ermöglicht die Bereitstellung von Arbeitslastclustern, die standardmäßig sicher sind.

vSphere IaaS control plane ist ein Add-On-Modul für vSphere, das die in vCenter Server und ESXi integrierten Sicherheitsfunktionen nutzen kann. Weitere Informationen finden Sie in der Dokumentation zu vSphere-Sicherheit .

Supervisor verschlüsselt alle geheimen Schlüssel, die in der Datenbank (etcd) gespeichert werden. Die geheimen Schlüssel werden über eine lokale Verschlüsselungsschlüsseldatei verschlüsselt, die beim Start von vCenter Server bereitgestellt wird. Der Entschlüsselungsschlüssel wird im Arbeitsspeicher (tempfs) auf den Supervisor-Knoten der Steuerungsebene und auf der Festplatte in verschlüsselter Form innerhalb der vCenter Server-Datenbank gespeichert. Der Schlüssel steht den Root-Benutzern jedes Systems als Klartext zur Verfügung.

Dasselbe Verschlüsselungsmodell gilt für die Daten in der Datenbank (etcd), die auf jeder TKG-Cluster-Steuerungsebene installiert ist. Alle etcd-Verbindungen werden mit Zertifikaten authentifiziert, die bei der Installation generiert und während Upgrades rotiert werden. Eine manuelle Rotation oder Aktualisierung der Zertifikate ist derzeit nicht möglich. Die in der Datenbank befindlichen geheimen Schlüssel aller Arbeitslastcluster werden in Klartext gespeichert.

Ein TKG-Cluster verfügt nicht über Infrastrukturanmeldedaten. Die Anmeldedaten, die in einem TKG-Cluster gespeichert werden, reichen nur für den Zugriff auf den vSphere-Namespace aus, in dem der TKG-Cluster mandantenfähig ist. Infolgedessen gibt es für Clusteroperatoren oder Benutzer keinen Eskalationsweg für Rechte.

Das für den Zugriff auf einen TKG-Cluster verwendete Authentifizierungstoken wird so skaliert, dass das Token nicht für den Zugriff auf den Supervisor oder andere TKG-Cluster genutzt werden kann. Dadurch wird verhindert, dass Clusteroperatoren oder Personen, die möglicherweise versuchen, einen Cluster zu kompromittieren, ihren Zugriff auf Root-Ebene nutzen, um bei der Anmeldung bei einem TKG-Cluster das Token eines vSphere-Administrators zu erfassen.

Ein TKG-Cluster ist standardmäßig sicher. Ab Tanzu Kubernetes-Version v1.25 ist für TKG-Cluster standardmäßig die Zugangssteuerung „Pod-Sicherheit“ (Pods Security Admission, PSA) aktiviert. Für Tanzu Kubernetes-Versionen bis v1.24 ist für jeden TKG-Cluster eine restriktive Pod-Sicherheitsrichtlinie (Pod Security Policy, PSP) verfügbar. Wenn Entwickler berechtigte Pods oder Root-Container ausführen müssen, muss ein Cluster-Administrator mindestens ein RoleBinding-Objekt erstellen, das dem Benutzer Zugriff auf die berechtigten Standard-PSP gewährt.

Konfigurieren von PSA für TKR 1.25 und höher

Ab den Tanzu Kubernetes Releases v1.25 und höher wird der PSA-Controller (Pod Security Admission, Zugangssteuerung für Pod-Sicherheit) aktiviert. Mit PSA können Sie Pod-Sicherheit unter Verwendung von Namespace-Bezeichnungen einheitlich erzwingen.

PSA in TKR 1.25 und höher aktiviert

Bei der Zugangssteuerung für Pod-Sicherheit handelt es sich um einen Kubernetes-Controller, mit dem Sie Sicherheitsstandards auf Pods anwenden können, die auf TKG-Clustern ausgeführt werden. Standardmäßig wird ab den Tanzu Kubernetes Releases v1.25 und höher der PSA-Controller (Pod Security Admission, Zugangssteuerung für Pod-Sicherheit) aktiviert. Der PSA-Controller ersetzt den veralteten PSP-Controller (Pod Security Policy, Pod-Sicherheitsrichtlinie), der entfernt wurde. Siehe auch Konfigurieren von PSP für TKR 1.24 und früher

Bei Tanzu Kubernetes Release v1.25 handelt es sich um eine Übergangsversion, bei der PSA so konfiguriert ist, dass eine Warnung erfolgt. Ab Tanzu Kubernetes Release v1.26 wird PSA erzwungen. Sie sollten die Migration von Pod-Arbeitslasten von PSP zu PSA im Voraus planen, um ein Upgrade der TKG-Cluster durchzuführen. Weitere Informationen finden Sie unter Migrieren von der Pod-Sicherheitsrichtlinie zur integrierten Zugangssteuerung für Pod-Sicherheit.

Clusterweite Konfiguration von PSA

Ab vSphere 8 Update 3 können Sie PSA clusterweit mithilfe der ClusterClass-Variable `podSecurityStandard` konfigurieren, die mit der v1beta1-API verfügbar ist. Weitere Informationen hierzu finden Sie unter Cluster-API v1beta1 .

PSA-Modi

Der PSA-Controller unterstützt drei Pod-Sicherheitsmodi: `enforce`, `audit` und `warn`. In der Tabelle werden alle PSA-Modi aufgelistet und beschrieben.

Tabelle 18-1. PSA-Modi

MODUS	Beschreibung
`enforce`	Sicherheitsverstöße führen dazu, dass der Pod abgelehnt wird.
`audit`	Sicherheitsverstöße lösen das Hinzufügen einer Überwachungsanmerkung zu dem im Überwachungsprotokoll aufgezeichneten Ereignis aus, sind aber ansonsten zulässig.
`warn`	Sicherheitsverstöße lösen eine benutzerseitige Warnung aus, sind aber ansonsten zulässig.

PSA-Standards

Der PSA-Controller definiert drei Ebenen von Pod-Sicherheitsstandards, die das Sicherheitsspektrum abdecken sollen. Die Pod-Sicherheitsstandards sind kumulativ und dabei permissiv bis restriktiv. In der Tabelle werden alle PSA-Standards aufgelistet und beschrieben.

Tabelle 18-2. PSA-Standards

EBENE	Beschreibung
`privileged`	Uneingeschränkte Steuerung, die die größtmögliche Berechtigungsstufe bietet. Diese Sicherheitsrichtlinie ermöglicht bekannte Berechtigungseskalationen.
`baseline`	Minimal restriktives Steuerelement, das bekannte Berechtigungseskalationen verhindert. Dieser Sicherheitsstandard lässt die standardmäßige (minimal angegebene) Pod-Konfiguration zu.
`restricted`	Stark eingeschränktes Steuerelement gemäß den aktuellen Best Practices für die Pod-Härtung.

PSA-Namespace-Bezeichnungen

Der PSA-Controller erzwingt Pod-Sicherheit auf Kubernetes-Namespace-Ebene. Mithilfe von Namespace-Bezeichnungen definieren Sie die PSA-Modi und -Ebenen, die Sie für Pods in einem bestimmten Namespace verwenden möchten.

Kubernetes bietet einen Satz von Bezeichnungen, mit denen Sie definieren können, welche der Standards für einen Namespace verwendet werden sollen. Die von Ihnen angewendete Bezeichnung definiert die von der Kubernetes-Steuerungsebene durchgeführte Aktion bei Erkennung eines potenziellen Verstoßes. Sie können für einen bestimmten Kubernetes-Namespace einen beliebigen oder alle Modi konfigurieren oder eine andere Ebene für verschiedene Modi festlegen.

Die Syntax der PSA-Namespace-Bezeichnung lautet wie folgt:

```
# MODE must be one of `enforce`, `audit`, or `warn`.
# LEVEL must be one of `privileged`, `baseline`, or `restricted`.
pod-security.kubernetes.io/<MODE>=<LEVEL>
```

Sie können auch eine Versionsbezeichnung pro Modus anwenden, die zum Anheften des Sicherheitsstandards an die Kubernetes-Version verwendet werden kann. Weitere Informationen finden Sie unter Erzwingen von Pod-Sicherheitsstandards mit Namespace-Bezeichnungen.

Standard-PSA für TKG-Cluster

Standardmäßig sind in TKG-Clustern, die mit Tanzu Kubernetes Release v1.25 bereitgestellt werden, die PSA-Modi `warn` und `audit` auf `restricted` für systemfremde Namespaces festgelegt. Dies ist eine Einstellung ohne zwangsweise Durchsetzung: Der PSA-Controller erzeugt eine Warn- und Überwachungsbenachrichtigung, wenn ein Pod gegen Sicherheitsrichtlinien verstößt, aber der Pod wird nicht abgelehnt.

Standardmäßig ist in TKG-Clustern, die mit Tanzu Kubernetes Releases v1.26 und höher bereitgestellt werden, der PSA-Modus `enforce` auf `restricted` für systemfremde Namespaces festgelegt. Wenn ein Pod gegen Sicherheitsrichtlinien verstößt, wird er abgelehnt. Sie müssen PSA im Namespace konfigurieren, um Pods weniger restriktiv auszuführen.

Wichtig Bestimmte System-Pods, die in den Namespaces „kube-system“, „tkg-system“ und „vmware-system-cloud-provider“ ausgeführt werden, benötigen erweiterte Berechtigungen. Diese Namespaces sind von der Pod-Sicherheit ausgeschlossen. Darüber hinaus kann Pod-Sicherheit in System-Namespaces nicht geändert werden.

In der Tabelle wird die PSA-Standardkonfiguration für TKG-Cluster aufgelistet:

Tabelle 18-3. Standard-PSA für TKG-Cluster

TKr-Version	Standard-PSA
TKr v1.25	Modus: Warnung | Ebene: eingeschränkt Modus: Überwachung | Ebene: eingeschränkt Modus: Erzwingung | Ebene: nicht festgelegt
TKr v1.26 und höher	Modus: Erzwingung | Ebene: eingeschränkt

Konfigurieren von PSA mithilfe von Namespace-Bezeichnungen

Verwenden Sie für Tanzu Kubernetes Release v1.25 den folgenden Beispielbefehl, um die Sicherheitsstufen für einen bestimmten Namespace so zu ändern, dass keine PSA-Warnungen und Überwachungsbenachrichtigungen erzeugt werden.

```
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/audit=privileged
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/warn=privileged
```

Verwenden Sie für Tanzu Kubernetes Releases v1.26 und höher den folgenden Beispielbefehl, um den PSA-Standard von `restricted` auf `baseline` herabzustufen.

```
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/enforce=baseline
```

So erzwingen Sie beispielsweise den `baseline`-Standard für den Standard-Namespace:

```
kubectl label --overwrite ns default pod-security.kubernetes.io/enforce=baseline
```

Verwenden Sie für Tanzu Kubernetes Releases v1.26 und höher den folgenden Beispielbefehl, um den PSA-Standard von `restricted` auf `privileged` herabzustufen.

```
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/enforce=privileged
```

So erzwingen Sie beispielsweise den `privileged`-Standard für den Standard-Namespace:

```
kubectl label --overwrite ns default pod-security.kubernetes.io/enforce=privileged
```

Verwenden Sie für Tanzu Kubernetes Releases v1.26 und höher die folgenden Beispielbefehle, um PSA über alle systemfremden Namespaces hinweg zu lockern.

```
kubectl label --overwrite ns --all pod-security.kubernetes.io/enforce=privileged
```

```
kubectl label --overwrite ns --all pod-security.kubernetes.io/warn=restricted
```

Konfigurieren des Sicherheitskontexts für einzelne Pods

Wenn Sie versuchen, einen Pod auszuführen, der gegen PSA verstößt, aktivieren Sie erneut eine Fehlermeldung, die darauf hinweist. Beispiel:

```
{"opType":"CREATE_POD","succeeded":false,"err":"creating pod example-pod: pods
\"example-pod\" is forbidden: violates PodSecurity \"restricted:latest\":
allowPrivilegeEscalation != false (container \"example-container\" must set
securityContext.allowPrivilegeEscalation=false), unrestricted capabilities
(container \"example-container\" must set securityContext.capabilities.drop=[\"ALL\"]),
runAsNonRoot != true (pod or container \"example-container\" must set
securityContext.runAsNonRoot=true),
seccompProfile (pod or container \"example-container\" must set
securityContext.seccompProfile.type to
\"RuntimeDefault\" or \"Localhost\")","events":[]}
```

Anstatt PSA für einen gesamten Namespace festzulegen, können Sie den Sicherheitskontext für einen einzelnen Pod konfigurieren. Damit ein einzelner Pod ausgeführt werden kann, können Sie in der Pod-Spezifikation den Sicherheitskontext wie folgt festlegen:

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
```

```
  - image: gcr.io/google_containers/busybox:1.24
    name: example-container
    command: ["/bin/sh", "-c", "echo 'hello' > /mnt/volume1/index.html && chmod
o+rX /mnt /mnt/volume1/index.html && while true ; do sleep 2 ; done"]
    securityContext:
      allowPrivilegeEscalation: false
      runAsNonRoot: true
      seccompProfile:
        type: "RuntimeDefault"
      capabilities:
        drop: [all]
    volumeMounts:
    - name: example-volume-mount
      mountPath: /mnt/volume1
  restartPolicy: Never
  volumes:
  - name: example-volume
    persistentVolumeClaim:
    claimName: example-pvc
```

Konfigurieren von PSP für TKR 1.24 und früher

TKG auf Supervisor unterstützt die Pod-Sicherheit mithilfe des Zugangscontrollers „Pod-Sicherheitsrichtlinie“, der standardmäßig für Tanzu Kubernetes-Cluster mit TKR v1.24 und früher aktiviert ist.

Voraussetzung: TKR 1.24 und früher

Der Inhalt dieses Themas gilt für TKG-Cluster auf Supervisor, die mit TKR v1.24 und früher bereitgestellt werden. Standardmäßig aktivieren diese Tanzu Kubernetes-Versionen die Zugangssteuerung für die Pod-Sicherheitsrichtlinie.

Der Nachfolger der Zugangssteuerung für die Pod-Sicherheitsrichtlinie ist die Zugangssteuerung „Pod-Sicherheit“. TKG-Cluster auf Supervisor, die mit TKR v1.25 und höher bereitgestellt wurden, aktivieren die Zugangssteuerung „Pod-Sicherheit“. Weitere Informationen finden Sie unter Konfigurieren von PSA für TKR 1.25 und höher.

Zugangssteuerung für Pod-Sicherheitsrichtlinie

Kubernetes-Pod-Sicherheitsrichtlinien (Pod Security Policies, PSPs) sind Ressourcen auf Clusterebene, die die Sicherheit der Pods kontrollieren. Mithilfe von PSPs haben Sie die Kontrolle darüber welche Pod-Typen von welchen Kontotypen bereitgestellt werden können.

Eine PodSecurityPolicy-Ressource definiert eine Reihe von Bedingungen, die ein Pod erfüllen muss, damit er bereitgestellt werden kann. Wenn diese Bedingungen nicht erfüllt sind, kann der Pod nicht bereitgestellt werden. Eine einzelne PodSecurityPolicy muss einen Pod in seiner Gesamtheit validieren. Ein Pod kann nicht einige seiner Regeln in einer Richtlinie und andere in einer anderen Richtlinie haben. Weitere Informationen finden Sie in der Kubernetes-Dokumentation unter Pod Security Policies, RBAC.

Es gibt verschiedene Möglichkeiten, die Verwendung von Pod-Sicherheitsrichtlinien in Kubernetes zu implementieren. Der typische Ansatz ist die Verwendung von RBAC-Objekten (rollenbasierte Zugriffssteuerung, Role-Based Access Control). ClusterRole und ClusterRoleBinding werden clusterweit angewendet; Role und RoleBinding gelten für einen bestimmten Namespace. Wenn RoleBinding verwendet wird, können Pods nur im selben Namespace wie die Bindung ausgeführt werden. Weitere Informationen finden Sie in der Kubernetes-Dokumentation unter RBAC.

Es gibt zwei Methoden zum Erstellen von Kubernetes-Pods: direkt oder indirekt. Sie erstellen einen Pod direkt, indem Sie eine Pod-Spezifikation mit Ihrem Benutzerkonto bereitstellen. Sie erstellen einen Pod indirekt, indem Sie Ressourcen auf höherer Ebene definieren, z. B. eine Bereitstellung oder ein DaemonSet. In diesem Fall erstellt ein Dienstkonto den zugrunde liegenden Pod. Weitere Informationen finden Sie in der Kubernetes-Dokumentation unter Service Accounts.

Um PSPs effektiv zu verwenden, müssen Sie beide Workflows zum Erstellen von Pods berücksichtigen: direkt und indirekt. Wenn ein Benutzer einen Pod direkt erstellt, steuert die an das Benutzerkonto gebundene PSP den Vorgang. Wenn ein Benutzer einen Pod mithilfe eines Dienstkontos erstellt, muss die PSP an das zum Erstellen des Pods verwendete Dienstkonto gebunden werden. Wenn in der Pod-Spezifikation kein Dienstkonto angegeben ist, wird das standardmäßige Dienstkonto für den Namespace verwendet.

PodSecurityPolicy-Standardressource für TKG-Cluster

In der Tabelle werden die berechtigten und beschränkten standardmäßigen Pod-Sicherheitsrichtlinien für TKG-Cluster und die mit den einzelnen Richtlinien verbundenen ClusterRole-Standardobjekte aufgelistet und beschrieben.

Tabelle 18-4. PodSecurityPolicy-Standardressource mit zugeordnetem ClusterRole-Objekt

Standard-PSP	Berechtigung	Beschreibung	Zugeordnetes ClusterRole-Standardobjekt
`vmware-system-privileged`	**Als beliebiger Benutzer ausführen**	Uneingeschränkte PSP. Entspricht der Ausführung eines Clusters ohne aktivierten PSP-Zugangscontroller.	`psp:vmware-system-privileged` kann diese PSP verwenden.
`vmware-system-restricted`	**Muss als Nicht-Root-Benutzer ausgeführt werden**	Eingeschränkte PSP. Erlaubt keinen berechtigten Zugriff auf Pod-Container, blockiert mögliche Eskalationen zu Root und erfordert die Nutzung mehrerer Sicherheitsmechanismen.	`psp:vmware-system-restricted` kann diese PSP verwenden.

Rollen- und ClusterRole-Bindungen für TKG-Cluster

TKG auf Supervisor stellt keine standardmäßigen RoleBinding- und ClusterRoleBinding-Objekte für TKG-Cluster zur Verfügung. Beispiele, die Sie verwenden können, finden Sie in dieser Dokumentation. Weitere Informationen finden Sie unter Anwenden der standardmäßigen Pod-Sicherheitsrichtlinie auf TKG-Dienstcluster.

Ein vCenter Single Sign-On-Benutzer, dem die Berechtigung **Bearbeiten** für einen vSphere-Namespace gewährt wird, wird der Rolle **cluster-admin** für jeden in diesem Namespace bereitgestellten Tanzu Kubernetes-Cluster zugeordnet. Ein authentifizierter Clusteradministrator kann implizit die PSP `vmware-system-privileged` verwenden. Obwohl es sich technisch nicht um ein ClusterRoleBinding-Objekt handelt, hat es dieselbe Wirkung.

Der Clusteradministrator muss beliebige Bindungen definieren, um die Pod-Typen, die Benutzer in einem Cluster bereitstellen können, zu erlauben oder einzuschränken. Wenn ein RoleBinding-Objekt verwendet wird, lässt die Bindung nur zu, dass Pods im selben Namespace wie die Bindung ausgeführt werden. Hierbei ist eine Kopplung mit Systemgruppen möglich, um Zugriff auf alle im Namespace ausgeführten Pods zu gewähren. Benutzer ohne Administratorrechte, die sich gegenüber dem Cluster authentifizieren, werden der Rolle `authenticated` zugewiesen und können als solche an die standardmäßige PSP gebunden werden.

Das folgende Verhalten wird für jeden TKG-Cluster erzwungen, der eine Pod-Sicherheitsrichtlinie verlangt:

- Ein Clusteradministrator kann berechtigte Pods mit seinem Benutzerkonto direkt in jedem Namespace erstellen.
- Ein Clusteradministrator kann im Namespace „kube-system“ Bereitstellungen, StatefulSet- und DaemonSet-Objekte erstellen (von denen jedes wiederum berechtigte Pods erstellt). Wenn Sie einen anderen Kubernetes-Namespace verwenden möchten, erstellen Sie dafür ein RoleBinding-Objekt oder ein ClusterRoleBinding-Objekt.
- Ein Clusteradministrator kann im Namespace „kube-system“ seine eigenen PSPs erstellen (zusätzlich zu den beiden Standard-PSPs) und diese PSPs an alle Benutzer binden. Wenn Sie Ihre eigene PSP definieren, finden Sie weitere Informationen in der englischsprachigen Dokumentation zu Kubernetes unter Policy Order (Reihenfolge der Richtlinien).
- Authentifizierte Benutzer können erst berechtigte oder nicht berechtigte Pods erstellen, nachdem der Clusteradministrator die PSPs an die authentifizierten Benutzer gebunden hat.

Beispiele für Bindungen finden Sie unter Anwenden der standardmäßigen Pod-Sicherheitsrichtlinie auf TKG-Dienstcluster.

Anwenden der standardmäßigen Pod-Sicherheitsrichtlinie auf TKG-Dienstcluster

TKG-Dienstcluster mit TKR 1.24 und früher enthalten die standardmäßige Pod-Sicherheitsrichtlinie, zu der Sie für die Bereitstellung von berechtigten und eingeschränkten Arbeitslasten eine Bindung herstellen können.

Anwenden der standardmäßigen Pod-Sicherheitsrichtlinie mithilfe von Rollen- und ClusterRole-Bindungen

vSphere IaaS control plane bietet Konfigurieren von PSP für TKR 1.24 und früher, die Sie auf TKG-Cluster auf Supervisor mit TKR 1.24 und früher anwenden können. Sie können dies tun, indem Sie RoleBinding- und ClusterRoleBinding-Objekte erstellen, die auf die Konfigurieren von PSP für TKR 1.24 und früherverweisen.

Hinweis Informationen zum Erstellen Ihrer eigenen Pod-Sicherheitsrichtlinie finden Sie in der Dokumentation zu Kubernetes.

Ein RoleBinding gewährt Berechtigungen innerhalb eines bestimmten Namespace. Ein ClusterRoleBinding gewährt Berechtigungen clusterweit. Die Entscheidung, RoleBindings oder ClusterRoleBinding zu verwenden, hängt von Ihrem Anwendungsfall ab. Wenn Sie beispielsweise ClusterRoleBinding verwenden und die Subjekte für die Verwendung von `system:serviceaccounts:<namespace>` konfigurieren, können Sie eine Bindung an eine PSP herstellen, bevor der Namespace erstellt wird. Weitere Informationen finden Sie unter RoleBinding and ClusterRoleBinding in der Kubernetes-Dokumentation.

Die folgenden Abschnitte enthalten YAML- und CLI-Befehle zum Erstellen von RoleBinding- und ClusterRoleBinding-Objekten, die die Verwendung Konfigurieren von PSP für TKR 1.24 und früher autorisieren.

Beispiel 1: ClusterRoleBinding-Objekt zum Ausführen eines privilegierten Satzes von Arbeitslasten

Mit dem folgenden kubectl-Befehl wird ein ClusterRoleBinding-Objekt erstellt, das authentifizierten Benutzer den Zugriff für die Ausführung eines privilegierten Satzes von Arbeitslasten mithilfe des Standard-PSP `vmware-system-privileged` gewährt.

Warnung Durch deklarative und imperative Anwendung von Beispiel 1 wird die clusterweite Bereitstellung privilegierter Arbeitslasten ermöglicht. In Beispiel 1 werden tatsächlich systemeigene Sicherheitskontrollen deaktiviert. Deshalb sollte dieses Beispiel mit Vorsicht und unter vollständiger Kenntnis der Auswirkungen verwendet werden. Für strengere Sicherheit sollten Sie die Beispiele 2, 3 und 4 in Betracht ziehen.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: psp:privileged
rules:
- apiGroups: ['policy']
  resources: ['podsecuritypolicies']
  verbs:     ['use']
  resourceNames:
  - vmware-system-privileged
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
```

```
metadata:
  name: all:psp:privileged
roleRef:
  kind: ClusterRole
  name: psp:privileged
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  name: system:serviceaccounts
  apiGroup: rbac.authorization.k8s.io
```

Alternativ zur Anwendung von YAML können Sie folgenden `kubectl`-Befehl ausführen:

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding --
clusterrole=psp:vmware-system-privileged --group=system:authenticated
```

Beispiel 2: RoleBinding-Objekt zum Ausführen eines berechtigten Satzes von Arbeitslasten

Der folgende kubectl-Befehl erstellt ein RoleBinding-Objekt, das für die Ausführung eines berechtigten Satzes von Arbeitslasten mithilfe der Standard-PSP `vmware-system-privileged` Zugriff auf alle Dienstkonten im Standard-Namespace gewährt.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rolebinding-default-privileged-sa-ns_default
  namespace: default
roleRef:
  kind: ClusterRole
  name: psp:vmware-system-privileged
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:serviceaccounts
```

Alternativ zur Anwendung von YAML können Sie folgenden kubectl-Befehl ausführen:

```
kubectl create rolebinding rolebinding-default-privileged-sa-ns_default --namespace=default --
clusterrole=psp:vmware-system-privileged --group=system:serviceaccounts
```

Beispiel 3: ClusterRoleBinding-Objekt zum Ausführen eines eingeschränkten Satzes von Arbeitslasten

Die folgende YAML erstellt ein ClusterRoleBinding-Objekt, das authentifizierten Benutzern für die Ausführung eines begrenzten Satzes von Arbeitslasten mithilfe der Standard-PSP `vmware-system-restricted` clusterweiten Zugriff gewährt.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: psp:authenticated
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:authenticated
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: psp:vmware-system-restricted
```

Alternativ zur Anwendung von YAML können Sie folgenden kubectl-Befehl ausführen:

```
kubectl create clusterrolebinding psp:authenticated --clusterrole=psp:vmware-system-
restricted --group=system:authenticated
```

Beispiel 4: RoleBinding-Objekt zum Ausführen eines eingeschränkten Satzes von Arbeitslasten

Die folgende YAML erstellt ein RoleBinding-Objekt, das für die Ausführung eines begrenzten Satzes von Arbeitslasten mithilfe der Standard-PSP `vmware-system-restricted` Zugriff auf alle Dienstkonten in einem bestimmten Namespace gewährt.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: psp:serviceaccounts
  namespace: some-namespace
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: psp:vmware-system-restricted
```

Alternativ zur Anwendung von YAML können Sie folgenden kubectl-Befehl ausführen:

Hinweis Ändern von „some-namespace“ in den Ziel-Namespace.

```
kubectl create rolebinding psp:serviceaccounts --clusterrole=psp:vmware-system-restricted --
group=system:serviceaccounts -n some-namespace
```

Beispielrolle für die Pod-Sicherheitsrichtlinie

Wenn Sie Ihre eigene Pod-Sicherheitsrichtlinie (PSP) für die Bereitstellung von Arbeitslasten definieren, verwenden Sie dieses Beispiel, um eine Rolle oder ClusterRole zu erstellen, die auf die benutzerdefinierte PSP verweist.

Das Beispiel veranschaulicht eine Rolle, die an eine PodSecurityPolicy gebunden ist. In der Rollendefinition wird der `example-role` das Verb `use` für eine von Ihnen definierte benutzerdefinierte PSP-Ressource gewährt. Alternativ können Sie eine der Standard-PSPs verwenden. Erstellen Sie dann eine Bindung.

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: Role
metadata:
  name: example-role
  namespace: tkgs-cluster-ns
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - create
  - get
  - list
  - watch
  - update
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - update
  - patch
- apiGroups:
  - extensions
  resourceNames:
  - CUSTOM-OR-DEFAULT-PSP
  resources:
  - podsecuritypolicies
  verbs:
  - use
```

Verwalten von TLS-Zertifikaten für TKG-Dienstcluster

vSphere IaaS control plane verwendet TLS-Verschlüsselung (Transport Layer Security), um die Kommunikation zwischen Komponenten zu sichern. TKG auf Supervisor enthält mehrere TLS-Zertifikate, die diese kryptografische Infrastruktur unterstützen. Die Supervisor-Zertifikatrotation erfolgt manuell. Die TKG-Zertifikatrotation erfolgt automatisiert, kann aber bei Bedarf manuell durchgeführt werden.

Informationen zu TLS-Zertifikaten für TKG-Dienstcluster

vSphere IaaS control plane verwendet TLS-Zertifikate zur Sicherung der Kommunikation zwischen den folgenden Komponenten:

- vCenter Server
- Supervisor-Knoten der Steuerungsebene
- ESXi-Hosts funktionieren als Worker-Knoten für vSphere-Pods
- TKG-Clusterknoten, sowohl Steuerungsebene als auch Worker

vSphere IaaS control plane arbeitet in den folgenden vertrauenswürdigen Domänen, um Systembenutzer zu authentifizieren.

Vertrauenswürdige Domäne	Beschreibung
Vertrauenswürdige Domäne von vCenter	Der Standardsignierer für TLS-Zertifikate in dieser vertrauenswürdigen Domäne ist die in vCenter Server integrierte VMware Certificate Authority (VMCA).
Vertrauenswürdige Domäne von Kubernetes	Der Standardsignierer für TLS-Zertifikate in dieser vertrauenswürdigen Domäne ist die Kubernetes-Zertifizierungsstelle (CA).

Weitere Informationen und vollständige Listen der einzelnen TLS-Zertifikate, die in der vSphere IaaS control plane-Umgebung verwendet werden, finden Sie im KB-Artikel vSphere with Tanzu Certificate Guide.

TLS-Zertifikatrotation

Das Verfahren zum Rotieren von TLS-Zertifikaten unterscheidet sich je nachdem, ob die Zertifikate für Supervisor oder für TKG-Dienstcluster verwendet werden.

Supervisor-Zertifikatrotation

TLS-Zertifikate für Supervisor werden vom VMCA-Zertifikat abgeleitet. Weitere Informationen zu Supervisor Zertifikaten finden Sie im KB-Artikel vSphere with Tanzu Certificate Guide.

Die Zertifikatrotation für Supervisor erfolgt manuell. Im KB-Artikel Replace vSphere with Tanzu Supervisor Certificates finden Sie Anweisungen zum Ersetzen von Supervisor Zertifikaten mithilfe des WCP-Tools Cert Manager.

TKG 2.0-Cluster-Zertifikatrotation

In der Regel müssen Sie die TLS-Zertifikate für einen TKG-Cluster nicht manuell rotieren, da die TLS-Zertifikate beim Aktualisieren eines TKG-Clusters beim Registrierungs-Aktualisierungsvorgang die TLS-Zertifikate automatisch für Sie rotiert.

Wenn die TLS-Zertifikate für einen TKG-Cluster nicht abgelaufen sind und Sie diese Zertifikate manuell rotieren müssen, können Sie dies tun, indem Sie die Schritte im folgenden Abschnitt ausführen.

Manuelles Rotieren der TLS-Zertifikate für einen TKG-Dienstcluster

Diese Anweisungen setzen fortgeschrittene Kenntnisse und Erfahrungen mit der TKG-Clusterverwaltung voraus. Diese Anweisungen setzen darüber hinaus voraus, dass die TLS-Zertifikate nicht abgelaufen sind. Wenn die Zertifikate abgelaufen sind, führen Sie diese Schritte nicht aus.

1 Melden Sie sich per SSH bei einem der Supervisor-Knoten an, um diese Schritte auszuführen. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu TKG-Dienst-Clustern als Kubernetes-Administrator und Systembenutzer.

2 Rufen Sie den Namen des TKG-Clusters ab.

```
export CLUSTER_NAMESPACE="tkg-cluster-ns"

kubectl get clusters -n $CLUSTER_NAMESPACE
NAME                    PHASE         AGE   VERSION
tkg-cluster             Provisioned   43h
```

3 Rufen Sie die kubeconfig-Datei des TKG-Clusters ab.

```
export CLUSTER_NAME="tkg-cluster"

kubectl get secrets -n $CLUSTER_NAMESPACE $CLUSTER_NAME-kubeconfig -o
jsonpath='{.data.value}' | base64 -d > $CLUSTER_NAME-kubeconfig
```

4 Rufen Sie den SSH-Schlüssel des TKG-Clusters ab.

```
kubectl get secrets -n $CLUSTER_NAMESPACE $CLUSTER_NAME-ssh -o jsonpath='{.data.ssh-
privatekey}' | base64 -d > $CLUSTER_NAME-ssh-privatekey
chmod 600 $CLUSTER_NAME-ssh-privatekey
```

5 Überprüfen Sie vor der Zertifikatrotation die Umgebung.

```
export KUBECONFIG=$CLUSTER_NAME-kubeconfig
```

```
kubectl get nodes -o wide
```

```
kubectl get nodes \
-o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}' \
-l node-role.kubernetes.io/master= > nodes
```

```
for i in `cat nodes`; do
    printf "\n######\n"
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
user@$i hostname
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
user@$i sudo kubeadm certs check-expiration
done;
```

Beispielergebnis der vorherigen Befehle:

```
tkg-cluster-control-plane-k8bqh
[check-expiration] Reading configuration from the cluster...
[check-expiration] FYI: You can look at this config file with 'kubectl -n kube-system get
cm kubeadm-config -o yaml'

CERTIFICATE                EXPIRES                  RESIDUAL TIME   CERTIFICATE
AUTHORITY   EXTERNALLY MANAGED
admin.conf                 Oct 04, 2023 23:00 UTC
363d                                  no
apiserver                  Oct 04, 2023 23:00 UTC   363d
ca                      no
apiserver-etcd-client      Oct 04, 2023 23:00 UTC   363d            etcd-
ca                 no
apiserver-kubelet-client   Oct 04, 2023 23:00 UTC   363d
ca                      no
controller-manager.conf    Oct 04, 2023 23:00 UTC
363d                                  no
etcd-healthcheck-client    Oct 04, 2023 23:00 UTC   363d            etcd-
ca                 no
etcd-peer                  Oct 04, 2023 23:00 UTC   363d            etcd-
ca                 no
etcd-server                Oct 04, 2023 23:00 UTC   363d            etcd-
ca                 no
front-proxy-client         Oct 04, 2023 23:00 UTC   363d            front-proxy-
ca          no
scheduler.conf             Oct 04, 2023 23:00 UTC
363d                                  no

CERTIFICATE AUTHORITY   EXPIRES                  RESIDUAL TIME   EXTERNALLY MANAGED
ca                      Oct 01, 2032 22:56 UTC   9y              no
etcd-ca                 Oct 01, 2032 22:56 UTC   9y              no
front-proxy-ca          Oct 01, 2032 22:56 UTC   9y              no
```

6 Rotieren Sie die TLS-Zertifikate für den TKG 2.0-Cluster.

Führen Sie einen Kontextwechsel zurück zum Supervisor durch, bevor Sie mit den folgenden Schritten fortfahren.

```
unset KUBECONFIG
kubectl config current-context
kubernetes-admin@kubernetes
```

```
kubectl get kcp  -n $CLUSTER_NAMESPACE $CLUSTER_NAME-control-plane -o
jsonpath='{.apiVersion}{"\n"}'
controlplane.cluster.x-k8s.io/v1beta1
```

```
kubectl get kcp -n $CLUSTER_NAMESPACE $CLUSTER_NAME-control-plane
NAME                         CLUSTER       INITIALIZED   API SERVER AVAILABLE   REPLICAS
READY   UPDATED   UNAVAILABLE   AGE   VERSION
tkg-cluster-control-plane   tkg-cluster   true          true                   3
3       3         0             43h   v1.21.6+vmware.1
```

```
kubectl patch kcp $CLUSTER_NAME-control-plane -n $CLUSTER_NAMESPACE --type merge -p
"{\"spec\":{\"rolloutAfter\":\"`date +'%Y-%m-%dT%TZ'`\"}}"
kubeadmcontrolplane.controlplane.cluster.x-k8s.io/tkg-cluster-control-plane patched
```

Maschinen-Rollout gestartet:

```
kubectl get machines -n $CLUSTER_NAMESPACE
NAME                                                CLUSTER
NODENAME
PROVIDERID                                          PHASE          AGE   VERSION
tkg-cluster-control-plane-k8bqh                     tkg-cluster   tkg-cluster-control-plane-
k8bqh                    vsphere://420a2e04-cf75-9b43-f5b6-23ec4df612eb   Running
43h   v1.21.6+vmware.1
tkg-cluster-control-plane-l7hwd                     tkg-cluster   tkg-cluster-control-plane-
l7hwd                    vsphere://420a57cd-a1a0-fec6-a741-19909854feb6   Running
43h   v1.21.6+vmware.1
tkg-cluster-control-plane-mm6xj                     tkg-cluster   tkg-cluster-control-plane-
mm6xj                    vsphere://420a67c2-ce1c-aacc-4f4c-0564daad4efa   Running
43h   v1.21.6+vmware.1
tkg-cluster-control-plane-nqdv6                     tkg-
cluster
                  Provisioning   25s   v1.21.6+vmware.1
tkg-cluster-workers-v8575-59c6645b4-wvnlz           tkg-cluster   tkg-cluster-workers-
v8575-59c6645b4-wvnlz          vsphere://420aa071-9ac2-02ea-6530-eb59ceabf87b
Running        43h   v1.21.6+vmware.1
```

Maschinen-Rollout abgeschlossen:

```
kubectl get machines -n $CLUSTER_NAMESPACE
NAME                                                CLUSTER
NODENAME
PROVIDERID                                          PHASE     AGE   VERSION
tkg-cluster-control-plane-m9745                     tkg-cluster   tkg-cluster-control-plane-
m9745                    vsphere://420a5758-50c4-3172-7caf-0bbacaf882d3   Running   17m
```

```
v1.21.6+vmware.1
tkg-cluster-control-plane-nqdv6                  tkg-cluster   tkg-cluster-control-plane-
nqdv6                  vsphere://420ad908-00c2-4b9b-74d8-8d197442e767   Running   22m
v1.21.6+vmware.1
tkg-cluster-control-plane-wdmph                  tkg-cluster   tkg-cluster-control-plane-
wdmph                  vsphere://420af38a-f9f8-cb21-e05d-c1bcb6840a93   Running   10m
v1.21.6+vmware.1
tkg-cluster-workers-v8575-59c6645b4-wvnlz        tkg-cluster   tkg-cluster-workers-
v8575-59c6645b4-wvnlz        vsphere://420aa071-9ac2-02ea-6530-eb59ceabf87b   Running
43h   v1.21.6+vmware.1
```

7 Überprüfen Sie die manuelle Zertifikatrotation für den TKG 2.0-Cluster.

Führen Sie zur Überprüfung der Zertifikatrotation die folgenden Befehle aus:

```
export KUBECONFIG=$CLUSTER_NAME-kubeconfig

kubectl get nodes -o wide
NAME                                       STATUS   ROLES                  AGE
VERSION            INTERNAL-IP   EXTERNAL-IP   OS-IMAGE                KERNEL-
VERSION       CONTAINER-RUNTIME
tkg-cluster-control-plane-m9745            Ready    control-plane,master   15m
v1.21.6+vmware.1   10.244.0.55   <none>        VMware Photon OS/Linux   4.19.198-1.ph3-
esx   containerd://1.4.11
tkg-cluster-control-plane-nqdv6            Ready    control-plane,master   21m
v1.21.6+vmware.1   10.244.0.54   <none>        VMware Photon OS/Linux   4.19.198-1.ph3-
esx   containerd://1.4.11
tkg-cluster-control-plane-wdmph            Ready    control-plane,master   9m22s
v1.21.6+vmware.1   10.244.0.56   <none>        VMware Photon OS/Linux   4.19.198-1.ph3-
esx   containerd://1.4.11
tkg-cluster-workers-v8575-59c6645b4-wvnlz  Ready    <none>                 43h
v1.21.6+vmware.1   10.244.0.51   <none>        VMware Photon OS/Linux   4.19.198-1.ph3-
esx   containerd://1.4.11

kubectl get nodes \
-o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}' \
-l node-role.kubernetes.io/master= > nodes

for i in `cat nodes`; do
    printf "\n######\n"
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
user@$i hostname
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
user@$i sudo kubeadm certs check-expiration
done;
```

Beispielergebnis mit den aktualisierten Ablaufdaten.

```
######
tkg-cluster-control-plane-m9745
[check-expiration] Reading configuration from the cluster...
[check-expiration] FYI: You can look at this config file with 'kubectl -n kube-system get
cm kubeadm-config -o yaml'

CERTIFICATE                EXPIRES                  RESIDUAL TIME   CERTIFICATE
```

```
AUTHORITY   EXTERNALLY MANAGED
admin.conf                 Oct 06, 2023 18:18 UTC
364d                                   no
apiserver                  Oct 06, 2023 18:18 UTC   364d
ca                      no
apiserver-etcd-client      Oct 06, 2023 18:18 UTC   364d            etcd-
ca                 no
apiserver-kubelet-client   Oct 06, 2023 18:18 UTC   364d
ca                      no
controller-manager.conf    Oct 06, 2023 18:18 UTC
364d                                   no
etcd-healthcheck-client    Oct 06, 2023 18:18 UTC   364d            etcd-
ca                 no
etcd-peer                  Oct 06, 2023 18:18 UTC   364d            etcd-
ca                 no
etcd-server                Oct 06, 2023 18:18 UTC   364d            etcd-
ca                 no
front-proxy-client         Oct 06, 2023 18:18 UTC   364d            front-proxy-
ca          no
scheduler.conf             Oct 06, 2023 18:18 UTC
364d                                   no

CERTIFICATE AUTHORITY   EXPIRES                  RESIDUAL TIME   EXTERNALLY MANAGED
ca                      Oct 01, 2032 22:56 UTC   9y              no
etcd-ca                 Oct 01, 2032 22:56 UTC   9y              no
front-proxy-ca          Oct 01, 2032 22:56 UTC   9y              no
```

8 Überprüfen Sie das Kubelet-Zertifikat.

Sie müssen das Kubelet-Zertifikat nicht rotieren, vorausgesetzt, der Parameter `rotateCertificates` in der kubelet-Konfiguration ist auf `true` festgelegt, was der Standardkonfiguration entspricht.

Diese Konfiguration kann mithilfe der folgenden Befehle überprüft werden:

```
kubectl get nodes \
-o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}' \
-l node-role.kubernetes.io/master!= > workernodes

for i in `cat workernodes`; do
    printf "\n######\n"
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
user@$i hostname
    ssh -o "StrictHostKeyChecking=no" -i $CLUSTER_NAME-ssh-privatekey -q vmware-system-
user@$i sudo grep rotate /var/lib/kubelet/config.yaml
done;
```

Beispielergebnis:

```
######
tkg-cluster-workers-v8575-59c6645b4-wvnlz
rotateCertificates: true
```

Rotieren von NSX-Zertifikaten

Supervisor verwendet TLS für die Kommunikation zwischen Supervisor und NSX. Es gibt verschiedene NSX Zertifikate, die Sie ggf. rotieren müssen, wenn Sie Supervisor mit dem NSX-Netzwerk-Stack bereitgestellt haben.

Von Supervisor verwendete NSX-Zertifikate

WCP mit NSX verwendet zwei Zertifikate für die Integration mit NSX:

- NSX Load Balancer-Zertifikat und -Schlüssel
- NSX Manager-Zertifikat und -Schlüssel

Weitere Informationen zu diesen Zertifikaten finden Sie im *Administratorhandbuch für NSX* unter Zertifikate für NSX-Verbund.

Hinweis Die Informationen in diesem Thema basieren auf NSX v3.2.

Rotieren des NSX Load Balancer-Zertifikats und -Schlüssels

Sie können das TLS-Zertifikat und den Schlüssel des NSX Load Balancer auf dem Bildschirm **Supervisor > Zertifikate > NSX Load Balancer** rotieren.

- Wählen Sie **Aktionen > CSR generieren** aus, um das Zertifikat zu generieren.
- Wählen Sie **Aktionen > Zertifikat ersetzen** aus, um das Zertifikat und den Schlüssel zu aktualisieren.

Generieren eines selbstsignierten Zertifikats und Schlüssels für jeden NSX Manager-Knoten

Supervisor verwendet das Enterprise-Administratorkonto, um auf die NSX Manager-API zuzugreifen. Wenn das NSX Manager-Zertifikat abläuft, kann Supervisor nicht auf NSX zugreifen.

Überprüfen Sie bei Ablauf von einem oder mehreren NSX Manager-Zertifikaten das Supervisor-Protokoll:

```
tail -f /var/log/vmware/wcp/wcpsvc.log
```

Möglicherweise werden dann Fehler ähnlich den folgenden angezeigt:

```
error wcp [kubelifecycle/nsx_pi.go:47] ... Error creating WCP service principal identity.
Err: NSX service-wide principal identity creation failed: ... x509: certificate has expired
```

```
error wcp [kubelifecycle/controller.go:554] ... Failed to create WCP service PI in NSX.
Err: WCP service principal idenitity creation failed: NSX service-wide principal identity
creation failed:
... x509: certificate has expired
```

Zum Beheben des Problems aktualisieren Sie das Zertifikat und den Schlüssel für jeden NSX Manager-Knoten. Wenn Sie einen NSX-Verwaltungscluster mit 3 Knoten mit einer VIP-Adresse verwenden, beachten Sie, dass Supervisor die VIP-Adresse nicht verwendet. Dies bedeutet, dass Sie jedes Zertifikat auf jedem NSX Manager-Knoten rotieren müssen. Sie können die Zertifikate nicht rotieren, indem Sie nur das VIP-Zertifikat ersetzen.

1 Um das Zertifikat für einen NSX Manager-Knoten zu rotieren, erstellen Sie eine Zertifikatsignieranforderung mit einem Namen und füllen Sie sie mit dem folgenden Inhalt auf.

   Dabei gilt:

   - `NSX-MGR-IP-ADDRESS` ist die NSX Manager-IP-Adresse
   - `NSX-MGR-FQDN` ist der NSX Manager-FQDN oder die IP-Adresse

   `nsx-mgr-01-cert.cnf`

```
[ req ]
default_bits = 2048
default_md = sha256
prompt = no
distinguished_name = req_distinguished_name
x509_extensions = SAN
req_extensions = v3_ca

[ req_distinguished_name ]
countryName = US
stateOrProvinceName = California
localityName = CA
organizationName = NSX
commonName = NSX-MGR-IP-ADDRESS  #CAN ONLY USE IF SAN IS ALSO USED

[ SAN ]
basicConstraints = CA:false
subjectKeyIdentifier = hash
authorityKeyIdentifier=keyid:always,issuer:always

[ v3_ca ]
subjectAltName = DNS:NSX-MGR-FQDN,IP:NSX-MGR-IP-ADDRESS  #MUST USE
```

   Beispiel:

```
[ req ]
default_bits = 2048
default_md = sha256
prompt = no
distinguished_name = req_distinguished_name
x509_extensions = SAN
req_extensions = v3_ca

[ req_distinguished_name ]
countryName = US
stateOrProvinceName = California
```

```
localityName = CA
organizationName = NSX
commonName = 10.197.79.122

[ SAN ]
basicConstraints = CA:false
subjectKeyIdentifier = hash
authorityKeyIdentifier=keyid:always,issuer:always

[ v3_ca ]
subjectAltName = DNS:10.197.79.122,IP:10.197.79.122
```

2 Verwenden Sie OpenSSL, um das SSL-Zertifikat und den privaten Schlüssel zu generieren.

```
openssl req -newkey rsa -nodes -days 1100 -x509 -config nsx-mgr-01-cert.cnf -keyout nsx-mgr-01.key -out nsx-mgr-01.crt
```

3 Stellen Sie sicher, dass die folgende Ausgabe angezeigt wird, nachdem Sie den Befehl ausgeführt haben.

```
Generating a 2048 bit RSA private key
...............+++
...............+++
writing new private key to 'nsx-mgr-01.key'
```

4 Es sollten drei Dateien angezeigt werden: die anfängliche Signieranforderung sowie das Zertifikat und der private Schlüssel, die durch Ausführen der Signieranforderung generiert wurden.

5 Führen Sie den folgenden Befehl aus, um das Zertifikat und den privaten Schlüssel zu überprüfen.

```
openssl x509 -in nsx-mgr-01.crt -text -noout
```

6 Wenn Sie einen NSX-Verwaltungscluster mit mehreren Knoten verwenden, wiederholen Sie den Vorgang für jeden NSX Manager-Knoten. Ändern Sie die IP-Adresse und den FQDN in der Zertifikatsignieranforderung und den Namen der Ausgabedateien entsprechend.

Importieren des SSL-Zertifikats und des privaten Schlüssels in die NSX-T Management-Konsole

Importieren Sie jedes NSX Manager-Knotenzertifikat und jeden privaten Schlüssel in NSX, indem Sie die folgenden Schritte befolgen. Wenn Sie die Dateien `nsx.crt` und `nsx.key` lokal speichern, können Sie diese in NSX hochladen oder den Inhalt kopieren bzw. einfügen.

1 Melden Sie sich bei der NSX-Verwaltungskonsole an und navigieren Sie zur Seite **System > Zertifikate**.

2 Klicken Sie auf **Importieren > Zertifikat importieren**.

Hinweis Da Sie ein selbstsigniertes Zertifikat generiert haben, wählen Sie auf jeden Falls **Zertifikat importieren** und nicht **CA-Zertifikat importieren** aus.

3 Geben Sie einen aussagekräftigen **Namen** für das Paar bestehend aus Zertifikat und Schlüssel ein, z. B. `nsx-mgr-01-cert-and-key`.

4 Navigieren Sie zu der Zertifikatsdatei und wählen Sie diese aus. Sie können den Inhalt auch kopieren und einschließlich Kopfzeile und Fußzeile einfügen.

Beispiel:

```
-----BEGIN CERTIFICATE-----
MIID+zCCAuOgAwIBAgIUCfXaWxNwXvrEFQbt+Dvvp9C/UkIwDQYJKoZIhvcNAQEL
BQAwVTELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExCzAJBgNVBAcM
...
FGlnyT4vxpa2TxvXNTCuXPV9z0VtVBF2QpUJluGH7W1i2wUnApCCXhItcBkfve0f
pCi9YoRoUT8fuMBYo7sL
-----END CERTIFICATE-----
```

5 Navigieren Sie zum ausgewählten Schlüssel und wählen Sie diesen aus. Sie können den Inhalt auch kopieren und einschließlich Kopfzeile und Fußzeile einfügen.

Beispiel:

```
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQC5GNlUSYHa1p+E
XuOGAsIgiFxUnerRYNm2ARMqRb/xKK6R4xgZhBmpmikpE90vQibvouHqnL13owq7
...
OzbnwMCUI2TeY1iJNx3HNKUrdLvrr8CMh7Looe0L4/2j7ygew2x2C5m272SCJYs/
ly+bOXEYaH4/ORHbvvr0jQ==
-----END PRIVATE KEY-----
```

6 Wählen Sie für **Dienstzertifikat** die Option **Nein** aus.

7 Geben Sie eine Beschreibung für das Paar bestehend aus Zertifikat und Schlüssel ein, z. B. `Cert and Private Key for NSX Manager Node 1`.

8 Klicken Sie auf **Import**.

9 Wiederholen Sie den Vorgang für jedes NSX Manager-Zertifikat und Schlüsselpaar.

Registrieren des NSX Manager-Zertifikats mithilfe der NSX API

Nachdem Sie die Zertifikate und Schlüssel in NSX Manager hochgeladen haben, registrieren Sie diese mithilfe der NSX API. Weitere Informationen finden Sie im *Administratorhandbuch für NSX* unter Importieren und Ersetzen von Zertifikaten.

1 Wählen Sie in NSX Manager **System > Zertifikate** aus.

2 Wählen Sie in der Spalte ID die ID des zu registrierenden Zertifikats aus und kopieren Sie die Zertifikat-ID aus dem Popup-Fenster.

3 Führen Sie den folgenden API-Aufruf aus, um die Zertifikate auflisten. Rufen Sie die Zertifikatknoten-IDs für jedes zu aktualisierende Zertifikat ab.

```
GET https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates
```

4 Führen Sie den folgenden API-Aufruf aus, um das Zertifikat zu validieren.

```
GET https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates/<cert-id>?
action=validate
```

Beispiel:

```
https://10.19.92.133/api/v1/trust-management/certificates/070bae44-7548-45ff-
a884-578f079eb6d4?action=validate
```

5 Führen Sie den folgenden API-Aufruf aus, um das Zertifikat eines NSX Manager Knotens zu ersetzen:

```
POST https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates/<cert-id>?
action=apply_certificate&service_type=API&node_id=<node-id>
```

Beispiel:

```
POST https://10.19.92.133/api/v1/trust-management/certificates/070bae44-7548-45ff-
a884-578f079eb6d4?
action=apply_certificate&service_type=API&node_id=e61c7537-3090-4149-b2b6-19915c20504f
```

6 Wenn Sie einen NSX-Verwaltungscluster mit mehreren Knoten verwenden, wiederholen Sie den Vorgang zum Ersetzen von Zertifikaten für jeden NSX Manager-Knoten.

7 Wenn Sie fertig sind, löschen Sie jedes abgelaufene Zertifikat, das Sie ersetzt haben. Sie können dies mithilfe der NSX Manager-Schnittstelle oder mithilfe der NSX API erledigen.

Beispiel:

```
https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates/<cert-id>
```

Integrieren von TMC in TKG-Dienst-Clustern

19

Dieser Abschnitt enthält Themen zur Integration von Tanzu Mission Control in TKG-Dienst-Cluster.

Lesen Sie als Nächstes die folgenden Themen:

- Registrieren von Tanzu Mission Control, die bei Supervisor gehostet werden
- Registrieren von Tanzu Mission Control Self-Managed bei Supervisor

Registrieren von Tanzu Mission Control, die bei Supervisor gehostet werden

Sie können Tanzu Kubernetes Grid auf Supervisor in Tanzu Mission Control integrieren. Dadurch können Sie TKG-Dienst-Cluster auf der gesamten TMC-Webschnittstelle verwalten.

Verwenden von TKG-Dienstclustern mit Tanzu Mission Control

Der allgemeine Workflow für die Verwendung von TKG-Dienst-Clustern mit Tanzu Mission Control und Tanzu Application Platform sieht wie folgt aus:

Schritt	Aktion
1	Bereitstellen eines TKG-Arbeitslastclusters Siehe Kapitel 7 Bereitstellen von TKG-Dienstclustern.
2	Registrieren von Supervisor bei TMC Siehe Registrieren von Supervisor bei Tanzu Mission Control.
3	Anhängen des TKG-Arbeitslastclusters an TMC Weitere Informationen finden Sie unter Verwalten des Lebenszyklus von Tanzu Kubernetes-Clustern.
4	Installieren von Tanzu Application Platform Weitere Informationen finden Sie in der Dokumentation zu Tanzu Application Platform.

Registrieren von Supervisor bei Tanzu Mission Control

Supervisor wird mit einem vSphere-Namespace für Tanzu Mission Control bereitgestellt. Um Supervisor bei Tanzu Mission Control zu registrieren, installieren Sie den TMC-Agent im vSphere-Namespace und führen den Registrierungsvorgang mithilfe von TMC durch.

Führen Sie dieses Verfahren durch, um den TMC-Agent auf Supervisor zu installieren. Beachten Sie, dass Sie bei diesem Verfahren den vSphere Client und die TMC-Webschnittstelle gleichzeitig verwenden müssen.

1 Authentifizieren Sie sich mithilfe des vSphere-Plug-In für kubectl beim Supervisor.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 Wechseln Sie in den Supervisor-Kontext, z. B.:

```
kubectl config use-context 10.199.95.59
```

3 Führen Sie den folgenden Befehl aus, um die Namespaces aufzulisten.

```
kubectl get ns
```

4 Der vSphere-Namespace für TMC heißt `svc-tmc-cXX` (wobei XX eine Zahl ist). Stellen Sie sicher, dass er vorhanden und aktiv ist.

5 Melden Sie sich bei der Tanzu Mission Control-Webschnittstelle an.

6 Registrieren Sie den Supervisor bei Tanzu Mission Control. Siehe Registrieren eines Verwaltungsclusters bei Tanzu Mission Control.

7 Kopieren Sie die Registrierungs-URL für Supervisor, die auf der Seite **Administration > Verwaltungscluster** verfügbar ist, über die Tanzu Mission Control-Webschnittstelle.

8 Öffnen Sie bei Bedarf einen Firewallport auf Supervisor für den Port, der von Tanzu Mission Control benötigt wird (in der Regel `443`). Weitere Informationen hierzu finden Sie unter Von den Cluster-Agent-Erweiterungen hergestellte ausgehende Verbindungen.

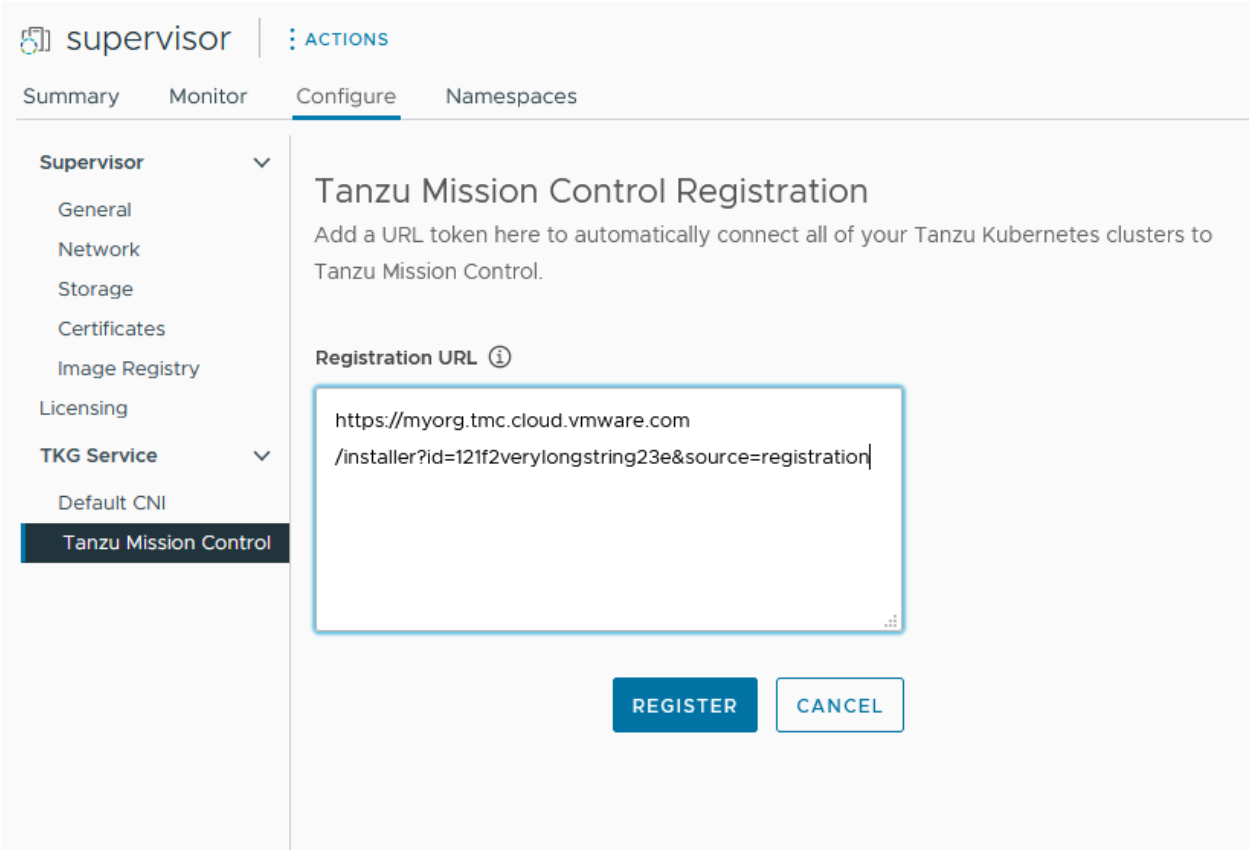
9 Melden Sie sich mit dem vSphere Client bei Ihrer vSphere IaaS control plane-Umgebung an.

10 Wählen Sie **Arbeitslastverwaltung** und den Ziel-Supervisor aus.

11 Wählen Sie **Konfigurieren** und dann **TKG-Dienst > Tanzu Mission Control** aus.

12 Geben Sie die Registrierungs-URL in das Feld **Registrierungs-URL** ein.

13 Klicken Sie auf **Registrieren**.



Sobald der Supervisor bei TMC registriert ist, verwenden Sie die TMC-Webschnittstelle, um TKG-Cluster bereitzustellen und zu verwalten. Anweisungen finden Sie in der Tanzu Mission Control-Dokumentation.

Deinstallieren des Tanzu Mission Control-Agents von Supervisor

Informationen zum Deinstallieren des Tanzu Mission Control-Agents über den Supervisor Sie unter Manuelles Entfernen des Cluster-Agents aus einem Supervisor-Cluster in vSphere mit Tanzu .

Registrieren von Tanzu Mission Control Self-Managed bei Supervisor

Um Tanzu Mission Control Self-Managed bei Supervisor zu registrieren, erstellen Sie eine benutzerdefinierte Ressourcendefinition für den TMC-Agent und wenden Sie sie an.

Über Tanzu Mission Control Self-Managed

Weitere Informationen zu Tanzu Mission Control Self-Managed, einschließlich der Installation und Konfiguration von, finden Sie in der Dokumentation Installieren und Ausführen von VMware Tanzu Mission Control Self-Managed.

Registrieren von Tanzu Mission Control Self-Managed bei Supervisor

Um Tanzu Mission Control Self-Managed in Supervisor zu integrieren, erstellen Sie eine benutzerdefinierte Ressourcendefinition, die auf den TMC-Agent verweist. Supervisor enthält einen Kubernetes-Namespace für TMC, in dem der Agent installiert ist.

Führen Sie das folgende Verfahren aus.

1 Installieren Sie Tanzu Mission Control Self-Managed, wie in der Dokumentation beschrieben. Weitere Informationen finden Sie unter Installieren und Ausführen von VMware Tanzu Mission Control Self-Managed.

2 Greifen Sie mithilfe eines Webbrowsers auf die lokale Bereitstellung von Tanzu Mission Control Self-Managed zu.

3 Exportieren Sie das Stamm-CA-Zertifikat für die Installation von Tanzu Mission Control Self-Managed.

   - Wenn Sie eine bekannte Zertifizierungsstelle verwenden, klicken Sie auf das Schlosssymbol links neben der Adressleiste im Browser und zeigen Sie das Zertifikat an. Wenn Sie eine private Zertifizierungsstelle verwenden, klicken Sie auf die Schaltfläche "Nicht sicher" und zeigen Sie das Zertifikat an.
   - Wählen Sie im Dialogfeld „Zertifikat“ die Registerkarte `Details` und dann die Schaltfläche `Export` , um eine Kopie des CA-Zertifikats herunterzuladen.
   - Öffnen Sie die CA-Zertifikatsdatei mit dem Texteditor Ihrer Wahl, um auf den Inhalt der Zertifizierungsstellenzertifikate zuzugreifen.

4 Authentifizieren Sie sich mithilfe des vSphere-Plug-In für kubectl beim Supervisor.

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

5 Führen Sie den folgenden Befehl aus, um die verfügbaren Kubectl-Kontexte aufzulisten.

```
kubectl config get-contexts
```

6 Ändern Sie den Kontext in den Ziel-vSphere-Namespace, in dem der TKG-Cluster bereitgestellt wird, auf dem Tanzu Mission Control Self-Managed läuft.

```
kubectl config use-context VSPHERE-NAMESPACE
```

7 Führen Sie den folgenden Befehl aus, um die Kubernetes-Namespaces aufzulisten.

```
kubectl get ns
```

8 Der Kubernetes-Namespace auf Supervisor für TMC heißt `svc-tmc-cXXXX` (wobei XXXX eine Zahl ist). Beispielsweise `svc-tmc-c1208`. Stellen Sie sicher, dass dieser Kubernetes-Namespace vorhanden und aktiv ist.

9 Verwenden Sie einen Texteditor, um die benutzerdefinierte Ressourcendefinition mit dem Namen `agentconfig.yaml` zu erstellen. Diese Datei enthält den TMC-Namespace, den Hostnamen Ihrer selbstverwalteten TMC-Bereitstellung und die Inhalte der Zertifizierungsstellenzertifikate.

- Geben Sie den Namen des Kubernetes-Namespace für TMC in das Feld `namespace` ein.
- Geben Sie die CA-Zertifikate in die Felder `caCerts` ein.
- Geben Sie den TMC-Hostnamen in das Feld `allowedHostNames` ein.

```
apiVersion: "installers.tmc.cloud.vmware.com/v1alpha1"
kind: "AgentConfig"
metadata:
  name: "tmc-agent-config"
  namespace: "<namespace>"
spec:
  caCerts: |-
    -----BEGIN CERTIFICATE-----
    Certificate1
    -----END CERTIFICATE-----
    -----BEGIN CERTIFICATE-----
    Certificate2
    -----END CERTIFICATE-----
  allowedHostNames:
    - "google.com"
```

10 Wenden Sie die `AgentConfig`-YAML-Datei an, die Sie erstellt haben.

```
kubectl apply -f agentconfig.yaml
```

11 Schließen Sie den Registrierungsvorgang über die -Webschnittstelle für Ihre Installation von Tanzu Mission Control Self-Managed ab. Anweisungen finden Sie in der Tanzu Mission Control-Dokumentation.

Sichern und Wiederherstellen von TKG-Dienstclustern und -Arbeitslasten

20

Informationen zum Sichern und Wiederherstellen von TKG-Dienstclustern und -Arbeitslasten finden Sie in diesem Abschnitt.

Lesen Sie als Nächstes die folgenden Themen:

- Überlegungen zur Sicherung und Wiederherstellung von TKG-Dienst-Dienstclustern und -Arbeitslasten
- Sichern und Wiederherstellen von Arbeitslasten mithilfe des Velero-Plug-In für vSphere
- Sichern und Wiederherstellen von TKG-Cluster-Arbeitslasten Supervisor mit eigenständigem Velero und Restic
- Sichern und Wiederherstellen mithilfe von Velero mit CSI-Snapshot

Überlegungen zur Sicherung und Wiederherstellung von TKG-Dienst-Dienstclustern und -Arbeitslasten

In diesem Abschnitt finden Sie Überlegungen zur Sicherung und Wiederherstellung von Arbeitslasten, die in TKG-Dienst-Clustern ausgeführt werden.

Sichern und Wiederherstellen von TKG-Dienst-Clustern

Zum Sichern und Wiederherstellen von TKG-Dienstclustern sichern Sie die Supervisor-Datenbank. Auf diese Weise können Sie vSphere-Namespaces-Objekte und VMs des TKG-Clusterknotens wiederherstellen.

Sie aktivieren die Supervisor-Sicherung und -Wiederherstellung mithilfe der vCenter Server-Sicherungsfunktion, die über die vCenter Server-Verwaltungsschnittstelle verfügbar ist. Weitere Informationen finden Sie in der Veröffentlichung zur Sicherungswiederherstellung für vSphere IaaS control plane.

Hinweis Sie können die Supervisor-Sicherung nur zum Wiederherstellen von TKG-Clusterknoten-VMs verwenden. Sie können die Supervisor-Sicherung nicht zum Wiederherstellen von Arbeitslasten verwenden, die auf TKG-Clustern bereitgestellt werden. Sie müssen Arbeitslasten separat sichern und sie dann wiederherstellen, nachdem der Cluster wiederhergestellt wurde.

Sichern und Wiederherstellen von Arbeitslasten, die auf TKG-Dienst-Clustern ausgeführt werden

In der Tabelle werden die Optionen für das Sichern und Wiederherstellen von statusfreien und statusbehafteten Arbeitslasten behandelt, die in TKG-Clustern ausgeführt werden.

Hinweis Das Sichern und Wiederherstellen eines Kubernetes-Clusters mit eigenständigem Velero ermöglicht Portabilität. Wenn Sie also Clusterarbeitslasten in einem Kubernetes-Cluster wiederherstellen möchten, der nicht vom TKG-Dienst bereitgestellt wurde, sollten Sie eigenständiges Velero verwenden.

Szenario	Tool	Anmerkungen
Sichern Sie statusfreie und statusbehaftete Arbeitslasten in einem TKG-Cluster auf Supervisor und stellen Sie sie in einem TKG-Cluster auf Supervisor wieder her.	Velero-Plug-In für vSphere Weitere Informationen hierzu finden Sie unter Sichern und Wiederherstellen von Arbeitslasten mithilfe des Velero-Plug-In für vSphere.	Sowohl Kubernetes-Metadaten als auch persistente Volumes können gesichert und wiederhergestellt werden. Velero-Snapshots werden für dauerhafte Volumes mit statusbehafteten Anwendungen verwendet. Erfordert, dass das Velero-Plug-In für vSphere auch auf Supervisor installiert und konfiguriert ist.
Sichern Sie statusfreie und statusbehaftete Arbeitslasten in einem TKG-Cluster auf Supervisor und stellen Sie sie in einem konformen Kubernetes-Cluster wieder her.	Eigenständiges Velero und Restic Siehe Sichern und Wiederherstellen von TKG-Cluster-Arbeitslasten Supervisor mit eigenständigem Velero und Restic. .	Sowohl Kubernetes-Metadaten als auch persistente Volumes können gesichert und wiederhergestellt werden. Restic wird für dauerhafte Volumes mit statusbehafteten Anwendungen verwendet. Verwenden Sie dieses Verfahren, wenn Sie Portabilität benötigen.
Sichern Sie statusfreie und statusbehaftete Arbeitslasten in einem TKG-Cluster auf Supervisor und stellen Sie sie in einem konformen Kubernetes-Cluster wieder her.	Eigenständiges Velero mit CSI-Snapshots Siehe Sichern und Wiederherstellen mithilfe von Velero mit CSI-Snapshot.	Erfordert vSphere 8.0 U2 oder höher und TKr v1.26 oder höher für vSphere 8.0.

Sichern und Wiederherstellen von Arbeitslasten mithilfe des Velero-Plug-In für vSphere

Dieser Abschnitt enthält Themen zum Sichern und Wiederherstellen von Clusterarbeitslasten mithilfe des Velero-Plug-In für vSphere.

Installieren und Konfigurieren des Velero-Plug-In für vSphere auf einem TKG-Cluster

Mit dem Velero-Plug-In für vSphere können Sie Arbeitslasten, die in einem TKGS-Cluster ausgeführt werden, sichern und wiederherstellen, indem Sie das Velero-Plug-In für vSphere in dem betreffenden Cluster installieren.

Überblick

Das Velero-Plug-In für vSphere bietet eine Lösung zum Sichern und Wiederherstellen von TKGS-Clusterarbeitslasten. Für persistente Arbeitslasten können Sie mit dem Velero-Plug-In für vSphere Snapshots der persistenten Volumes erstellen.

Hinweis Wenn Sie Portabilität für die TKGS-Clusterarbeitslasten benötigen, die Sie sichern und wiederherstellen möchten, verwenden Sie nicht das Velero-Plug-In für vSphere. Verwenden Sie für die Kubernetes-Cluster-übergreifende Portabilität eigenständiges Velero mit Restic.

Voraussetzung: Installation des Velero-Plug-In für vSphere im Supervisor

Für die Installation des Velero-Plug-In für vSphere in einem TKGS-Cluster muss das Velero-Plug-In für vSphere im Supervisor installiert sein. Außerdem muss der Supervisor mit NSX-Netzwerk konfiguriert sein. Weitere Informationen finden Sie unter #unique_22.

Schritt 1: Installieren der Velero-CLI auf einem Linux-Workstation

Die Velero-CLI ist das Standardtool für die Schnittstelle mit Velero. Die Velero-CLI bietet mehr Funktionen als die Velero-Plug-In für vSphere-CLI (`velero-vsphere`) und ist für die Sicherung und Wiederherstellung von Tanzu Kubernetes-Cluster-Arbeitslasten erforderlich.

Installieren Sie die Velero-CLI auf einer Linux-Workstation. Idealerweise handelt es sich um denselben Jump-Host, in dem Sie zugehörige CLIs für Ihre vSphere IaaS control plane-Umgebung ausführen, einschließlich`kubectl`, `kubectl-vsphere`, und `velero-vsphere`.

Die Velero-Versionsnummern werden als `X.Y.Z` dargestellt. In der Velero-Kompatibilitätsmatrix finden Sie die spezifischen Versionen, die verwendet werden sollen. Ersetzen Sie sie entsprechend beim Ausführen der Befehle.

Führen Sie die folgenden Schritte aus, um die Velero-CLI zu installieren.

1 Führen Sie folgende Befehle aus:

```
$ wget https://github.com/vmware-tanzu/velero/releases/download/vX.Y.Z/velero-vX.Y.Z-linux-amd64.tar.gz
$ gzip -d velero-vX.Y.Z-linux-amd64.tar.gz && tar -xvf velero-vX.Y.Z-linux-amd64.tar
$ export PATH="$(pwd)/velero-vX.Y.Z-linux-amd64:$PATH"

$ which velero
/root/velero-vX.Y.Z-linux-amd64/velero
```

2 Überprüfen Sie die Installation der Velero-CLI.

```
velero version

Client:
    Version: vX.Y.Z
```

Schritt 2 : Abrufen der Details des S3-kompatiblen Buckets

Der Einfachheit halber wird in den Schritten davon ausgegangen, dass Sie denselben S3-kompatiblen Objektspeicher verwenden, den Sie beim Installieren des Velero-Plug-In für vSphere im Supervisor konfiguriert haben. In der Produktion können Sie einen separaten Objektspeicher erstellen.

Für die Installation des Velero-Plug-In für vSphere müssen Sie die folgenden Informationen über Ihren S3-kompatiblen Objektspeicher angeben.

Datenelement	Beispielwert
s3Url	`http://my-s3-store.example.com`
aws_access_key_id	ACCESS-KEY-ID-STRING
aws_secret_access_key	SECRET-ACCESS-KEY-STRING

Erstellen Sie eine Datei für geheime Schlüssel mit dem Namen `s3-credentials` und den folgenden Informationen. Beim Installieren des Velero-Plug-In für vSphere werden Sie auf diese Datei verweisen.

```
aws_access_key_id = ACCESS-KEY-ID-STRING
aws_secret_access_key = SECRET-ACCESS-KEY-STRING
```

Schritt 3 Option A: Installieren des Velero-Plug-In für vSphere auf dem TKG-Cluster mithilfe einer Bezeichnung (neue Methode)

Wenn Sie vSphere 8 Update 3 oder höher verwenden, können Sie das Velero-Plug-In für vSphere automatisch in einem TKG-Cluster installieren, indem Sie eine Bezeichnung hinzufügen.

1 Stellen Sie sicher, dass auf den Sicherungsspeicherort zugegriffen werden kann.

2 Stellen Sie sicher, dass der Velero vSphere-Operator-Kern Supervisor-Dienst aktiviert ist.

```
kubectl get ns | grep velero
svc-velero-domain-c9                        Active   18d
```

3 Stellen Sie sicher, dass ein Kubernetes-Namespace mit der Bezeichnung `velero` auf Supervisor erstellt wird.

```
kubectl get ns | grep velero
svc-velero-domain-c9                        Active   18d
velero                                      Active   1s
```

4 Stellen Sie sicher, dass das Velero-Plug-In für vSphere Supervisor-Dienst auf Supervisor aktiviert ist.

```
velero version
Client:
        Version: v1.11.1
        Git commit: bdbe7eb242b0f64d5b04a7fea86d1edbb3a3587c
Server:
        Version: v1.11.1
```

```
kubectl get veleroservice -A
NAMESPACE   NAME      AGE
velero      default   53m
```

```
velero backup-location get
NAME      PROVIDER   BUCKET/PREFIX   PHASE       LAST VALIDATED                  ACCESS
MODE   DEFAULT
default   aws        velero          Available   2023-11-20 14:10:57 -0800 PST
ReadWrite     true
```

5 Aktivieren Sie Velero für den TKG-Zielcluster, indem Sie die Velero-Bezeichnung zum Cluster hinzufügen.

```
kubectl label cluster CLUSTER-NAME --namespace CLUSTER-NS velero.vsphere.vmware.com/
enabled=true
```

Hinweis Die Aktivierung erfolgt über den vSphere-Namespace, wenn der Cluster bereitgestellt wird.

6 Stellen Sie sicher, dass Velero installiert ist und für den Cluster verwendet werden kann.

```
kubectl get ns
NAME                                STATUS   AGE
...
velero                              Active   2m    <--
velero-vsphere-plugin-backupdriver  Active   2d23h
```

```
kubectl get all -n velero
NAME                                READY   STATUS    RESTARTS   AGE
pod/backup-driver-5945d6bcd4-gtw9d  1/1     Running   0          17h
pod/velero-6b9b49449-pq6b4          1/1     Running   0          18h
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/backup-driver     1/1     1            1           17h
```

```
deployment.apps/velero          1/1     1            1           18h
NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/backup-driver-5945d6bcd4   1         1         1       17h
replicaset.apps/velero-6b9b49449           1         1         1       18h
```

```
velero version
Client:
        Version: v1.11.1
        Git commit: bdbe7eb242b0f64d5b04a7fea86d1edbb3a3587c
Server:
        Version: v1.11.1
```

Schritt 3 Option B: Manuelles Installieren des Velero-Plug-In für vSphere im TKG-Cluster (veraltete Methode)

Sie verwenden die Velero-CLI zum Installieren des Velero-Plug-In für vSphere im TKG-Zielcluster, den Sie sichern und wiederherstellen möchten.

Der Velero-CLI-Kontext folgt automatisch dem `kubectl`-Kontext. Stellen Sie vor der Ausführung von Velero-CLI-Befehlen zur Installation von Velero und des Velero-Plug-In für vSphere auf dem Zielcluster sicher, dass als `kubectl`-Kontext der Zielcluster festgelegt ist.

1 Authentifizieren Sie sich mithilfe des vSphere-Plug-In für kubectl beim Supervisor.

2 Ändern Sie den `kubectl`-Kontext in den TKG-Zielcluster.

```
kubectl config use-context TARGET-TANZU-KUBERNETES-CLUSTER
```

3 Erstellen Sie im TKG-Cluster eine Configmap für das Velero-Plug-In mit der Bezeichnung `velero-vsphere-plugin-config.yaml`.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: velero-vsphere-plugin-config
data:
  cluster_flavor: GUEST
```

Wenden Sie die Configmap auf den TKG-Cluster an.

```
kubectl apply -n <velero-namespace> -f velero-vsphere-plugin-config.yaml
```

Wenn Sie die Configmap nicht installieren, erhalten Sie folgende Fehlermeldung bei der Installation des Velero-Plug-In für vSphere.

```
Error received while retrieving cluster flavor from config, err: configmaps "velero-
vsphere-plugin-config" not found
Falling back to retrieving cluster flavor from vSphere CSI Driver Deployment
```

4 Führen Sie den folgenden Velero-CLI-Befehl aus, um Velero auf dem Zielcluster zu installieren.

Ersetzen Sie die Platzhalterwerte für die Felder **`BUCKET-NAME`**, **`REGION`** (zwei Instanzen) und **`s3Url`** durch die geeigneten Werte. Wenn Sie von einer der vorstehenden Anweisungen abgewichen sind, passen Sie auch diese Werte an, z. B. den Namen oder den Speicherort der Geheimschlüssel-Datei, den Namen des manuell erstellten `velero`-Namespace usw.

```
./velero install --provider aws \
--bucket BUCKET-NAME \
--secret-file ./s3-credentials \
--features=EnableVSphereItemActionPlugin \
--plugins velero/velero-plugin-for-aws:vX.Y.Z \
--snapshot-location-config region=REGION \
--backup-location-config region=REGION,s3ForcePathStyle="true",s3Url=http://my-s3-
store.example.com
```

5 Installieren Sie das Velero-Plug-In für vSphere auf dem Zielcluster. Der installierte Velero kommuniziert mit dem Kubernetes-API-Server, um das Plug-In zu installieren.

```
velero plugin add vsphereveleroplugin/velero-plugin-for-vsphere:vX.Y.Z
```

Ergänzung: Deinstallieren des Velero-Plug-In für vSphere im TKG-Cluster

Mit den folgenden Schritten deinstallieren Sie das Velero-Plug-In für vSphere.

1 Ändern Sie den `kubectl`-Kontext in den Tanzu Kubernetes-Zielcluster.

```
kubectl config use-context TARGET-TANZU-KUBERNETES-CLUSTER
```

2 Führen Sie zum Installieren des Plug-Ins den folgenden Befehl aus, um den InitContainer von velero-plugin-for-vsphere aus der Velero-Bereitstellung zu entfernen.

```
velero plugin remove vsphereveleroplugin/velero-plugin-for-vsphere:vX.Y.Z
```

3 Um den Vorgang abzuschließen, löschen Sie die Backup-Treiber-Bereitstellung und die zugehörigen CRDs.

```
kubectl -n velero delete deployment.apps/backup-driver
```

```
kubectl delete crds \
backuprepositories.backupdriver.cnsdp.vmware.com \
backuprepositoryclaims.backupdriver.cnsdp.vmware.com \
clonefromsnapshots.backupdriver.cnsdp.vmware.com \
deletesnapshots.backupdriver.cnsdp.vmware.com \
snapshots.backupdriver.cnsdp.vmware.com
```

```
kubectl delete crds uploads.datamover.cnsdp.vmware.com downloads.datamover.cnsdp.vmware.com
```

Sichern und Wiederherstellen von Arbeitslasten im TKG-Cluster mit dem Velero-Plug-In für vSphere

Sie können Arbeitslasten, die in TKG-Clustern auf Supervisor ausgeführt werden, mithilfe des Velero-Plug-In für vSphere sichern und wiederherstellen.

Voraussetzungen

Um TKG-Clusterarbeitslasten mit dem Velero-Plug-In für vSphere zu sichern und wiederherzustellen, müssen Sie zuerst das Velero-Plug-In für vSphere im Zielcluster installieren. Weitere Informationen finden Sie unter Installieren und Konfigurieren des Velero-Plug-In für vSphere auf einem TKG-Cluster.

Sichern einer Arbeitslast

Nachfolgend finden Sie einen Beispielbefehl zum Erstellen einer Velero-Sicherung.

```
velero backup create <backup name> --include-namespaces=my-namespace
```

Die Velero-Sicherung wird als `Completed` gekennzeichnet, nachdem alle lokalen Snapshots erstellt wurden und Kubernetes-Metadaten mit Ausnahme von Volume-Snapshots in den Objektspeicher hochgeladen wurden. Zu diesem Zeitpunkt werden asynchrone Datenverschiebungsaufgaben, d. h. das Hochladen von Volume-Snapshots, weiterhin im Hintergrund ausgeführt und können einige Zeit in Anspruch nehmen. Sie können den Status des Volume-Snapshots überprüfen, indem Sie benutzerdefinierte Snapshot-Ressourcen (Custom Resources, CRs) überwachen.

Snapshots

Snapshots werden zum Sichern persistenter Volumes verwendet. Für jeden Volume-Snapshot wird eine Snapshot-CR im selben Namespace erstellt wie die Anforderung eines dauerhaften Datenträgers (Persistent Volume Claim, PVC), für die ein Snapshot erstellt wird.

Sie können alle Snapshots im PVC-Namespace abrufen, indem Sie den folgenden Befehl ausführen.

```
kubectl get -n <pvc namespace> snapshot
```

Die Definition der benutzerdefinierten Snapshot-Ressource (Custom Resource Definition, CRD) verfügt über eine Reihe von Statusangaben (Phasen) für das Feld „`.status.phase`“, einschließlich:

Snapshot-Status	Beschreibung
Neu	Noch nicht verarbeitet
Snapshotted	Lokaler Snapshot wurde erstellt
SnapshotFailed	Lokaler Snapshot ist fehlgeschlagen
Upload wird durchgeführt ...	Der Snapshot wird hochgeladen
Uploaded	Der Snapshot wird hochgeladen
UploadFailed	Der Snapshot konnte nicht hochgeladen werden

Snapshot-Status	Beschreibung
Canceling	Das Hochladen des Snapshots wird abgebrochen
Abgebrochen	Das Hochladen des Snapshots wurde abgebrochen
CleanupAfterUploadFailed	Die Bereinigung des lokalen Snapshots nach dem Hochladen des Snapshots ist fehlgeschlagen

Wiederherstellen einer Arbeitslast

Nachfolgend finden Sie einen Beispielbefehl für die Velero-Wiederherstellung.

```
velero restore create --from-backup <velero-backup-name>
```

Die Velero-Wiederherstellung wird als `Completed` gekennzeichnet, wenn Volume-Snapshots und andere Kubernetes-Metadaten erfolgreich im aktuellen Cluster wiederhergestellt wurden. Zu diesem Zeitpunkt sind alle Aufgaben des vSphere Plug-Ins im Zusammenhang mit dieser Wiederherstellung ebenfalls abgeschlossen. Im Falle einer Velero-Sicherung werden keine asynchronen Datenverschiebungsaufgaben im Hintergrund ausgeführt.

CloneFromSnapshots

Um die Wiederherstellung für jeden Volume-Snapshot durchzuführen, wird eine benutzerdefinierte `CloneFromSnapshot`-Ressource (Custom Resource, CR) im selben Namespace erstellt wie die PVC, für die ursprünglich ein Snapshot erstellt wurde. Sie können alle CloneFromSnapshots im PVC-Namespace abrufen, indem Sie den folgenden Befehl ausführen.

```
kubectl -n <pvc namespace> get clonefromsnapshot
```

Die CloneFromSnapshot-CRD weist einige Schlüsselstatusangaben für das Feld „`.status.phase`“ auf:

Snapshot-Status	Beschreibung
Neu	Das Klonen aus dem Snapshot ist nicht abgeschlossen
InProgress	Der Snapshot des vSphere Volume wird aus dem Remote-Repository heruntergeladen
Abgeschlossen	Das Klonen aus dem Snapshot ist abgeschlossen
Fehlgeschlagen	Das Klonen aus dem Snapshot ist fehlgeschlagen

Sichern und Wiederherstellen von TKG-Cluster-Arbeitslasten Supervisor mit eigenständigem Velero und Restic

Dieser Abschnitt enthält Themen zum Sichern und Wiederherstellen von unter Supervisor ausgeführten TKG-Cluster-Arbeitslasten mit eigenständigem Velero und Restic.

Installieren und Konfigurieren von eigenständigem Velero und Restic in TKG-Clustern

Um Arbeitslasten, die in TKG-Clustern ausgeführt werden, auf Supervisor zu sichern und wiederherzustellen, erstellen Sie einen Datenspeicher und installieren Sie Velero mit Restic im Kubernetes-Cluster.

Überblick

TKG-Cluster werden auf VM-Knoten ausgeführt. Zum Sichern und Wiederherstellen von TKG-Clusterarbeitslasten installieren Sie Velero und Restic im Cluster.

Voraussetzungen

Achten Sie darauf, dass Ihre Umgebung die folgenden Voraussetzungen für die Installation von Velero und Restic zum Sichern und Wiederherstellen von Arbeitslasten auf Tanzu Kubernetes-Clustern erfüllt.

- Eine Linux-VM mit ausreichend Speicherplatz zum Speichern mehrerer Arbeitslastsicherungen. Sie installieren MinIO auf dieser VM.
- Eine Linux-VM, in der die Kubernetes-CLI-Tools für vSphere installiert sind, einschließlich des vSphere-Plug-In für kubectl und kubectl. Sie installieren die Velero-CLI auf dieser Client-VM. Wenn Sie über keine solche VM verfügen, können Sie die Velero CLI lokal installieren, müssen die Installationsschritte jedoch entsprechend anpassen.
- Die Kubernetes-Umgebung verfügt über Internetzugriff und kann von der Client-VM erreicht werden.

Installieren und Konfigurieren des MinIO-Objektspeichers

Velero benötigt einen S3-kompatiblen Objektspeicher als Ziel für Kubernetes-Arbeitslastsicherungen. Velero unterstützt mehrere solcher Objektspeicher-Anbieter. Der Einfachheit halber wird in dieser Anleitung MinIO verwendet. Dabei handelt es sich um einen S3-kompatiblen Speicherdienst, der lokal auf der Objektspeicher-VM ausgeführt wird.

1 Installieren Sie MinIO.

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
```

2 Erteilen Sie MinIO-Ausführungsberechtigungen.

```
chmod +x minio
```

3 Erstellen Sie ein Verzeichnis auf dem Dateisystem für MinIO.

```
mkdir /DATA-MINIO
```

4 Starten Sie den MinIO-Server.

```
./minio server /DATA-MINIO
```

5 Nach dem Start des MinIO-Servers erhalten Sie wichtige Datenspeicherinstanzdetails, darunter die Endpoint-URL, den AccessKey und den SecretKey. Zeichnen Sie die Endpoint-URL, den AccessKey und den SecretKey in der Tabelle auf.

Datenspeicher-Metadaten	Wert
Endpoint-URL	
AccessKey	
SecretKey	

6 Navigieren Sie zum MinIO-Datenspeicher, indem Sie einen Browser öffnen und darin die Endpoint-URL des MinIO-Servers aufrufen.

7 Melden Sie sich beim MinIO-Server an und geben Sie den AccessKey und SecretKey an.

8 Um MinIO als Dienst zu aktivieren, konfigurieren Sie MinIO für den automatischen Start, indem Sie das `minio.service`-Skript herunterladen.

```
curl -O https://raw.githubusercontent.com/minio/minio-service/master/linux-systemd/
minio.service
```

9 Bearbeiten Sie das `minio.service`-Skript und fügen Sie den folgenden Wert für `ExecStart` hinzu.

```
ExecStart=/usr/local/bin/minio server /DATA-MINIO path
```

10 Speichern Sie das überarbeitete Skript.

11 Konfigurieren Sie den MinIO-Dienst, indem Sie die folgenden Befehle ausführen.

```
cp minio.service /etc/systemd/system
cp minio /usr/local/bin/
systemctl daemon-reload
systemctl start minio
systemctl status minio
systemctl enable minio
```

12 Erstellen Sie einen MinIO-Bucket für die Sicherung und Wiederherstellung, indem Sie den MinIO-Browser starten und sich bei Ihrem Objektspeicher anmelden.

13 Klicken Sie auf das Symbol für „Bucket erstellen“.

14 Geben Sie den Bucket-Namen ein, z. B.: `my-cluster-backups`.

15 Überprüfen Sie, ob der Bucket erstellt wurde.

16 Standardmäßig ist ein neuer MinIO-Bucket schreibgeschützt. Für die Sicherung und Wiederherstellung mit eigenständigem Velero muss der MinIO-Bucket über Lese- und Schreibberechtigung verfügen. Um die Lese- und Schreibberechtigung für den Bucket zu aktivieren, wählen Sie den Bucket aus und klicken Sie auf den Link mit der Ellipse (...).

17 Klicken Sie auf **Richtlinie bearbeiten**.

18 Ändern Sie die Richtlinie in **Lesen und Schreiben**.

19 Klicken Sie auf **Hinzufügen**.

20 Klicken Sie auf das X, um das Dialogfeld zu schließen.

Installieren der Velero-CLI

Installieren Sie die Velero-CLI auf dem VM-Client oder auf Ihrem lokalen Computer.

Die für diese Dokumentation verwendete Version ist *Velero 1.9.7 für Tanzu Kubernetes Grid 2.2.0*.

1 Laden Sie Velero von der Tanzu Kubernetes Grid-Produkt-Download-Seite im VMware Customer Connect-Portal herunter.

Hinweis Sie müssen die von VMware signierte Velero-Binärdatei verwenden, um Unterstützung von VMware zu erhalten.

2 Öffnen Sie eine Befehlszeile und wechseln Sie zum Velero-CLI-Download.

3 Entpacken Sie die Download-Datei. Beispiel:

```
gunzip velero-linux-vX.X.X_vmware.1.gz
```

4 Überprüfen Sie, ob die Velero-Binärdatei vorhanden ist.

```
ls -l
```

5 Erteilen Sie der Velero-CLI Ausführungsberechtigungen.

```
chmod +x velero-linux-vX.X.X_vmware.1
```

6 Machen Sie die Velero-CLI global verfügbar, indem Sie sie in den Systempfad verschieben:

```
cp velero-linux-vX.X.X_vmware.1 /usr/local/bin/velero
```

7 Überprüfen Sie die Installation.

```
velero version
```

Installieren von Velero und Restic auf dem Tanzu Kubernetes-Cluster

Der Velero CLI-Kontext folgt automatisch dem kubectl-Kontext. Legen Sie den kubectl-Kontext fest, bevor Sie Velero-CLI-Befehle zum Installieren von Velero und Restic auf dem Zielcluster ausführen.

1 Rufen Sie den Namen des MinIO-Buckets ab. Beispiel: `my-cluster-backups`.

2 Rufen Sie den AccessKey und den SecretKey für den MinIO-Bucket ab.

3 Legen Sie den Kontext für den Kubernetes-Zielcluster so fest, dass die Velero-CLI weiß, an welchem Cluster gearbeitet werden soll.

```
kubectl config use-context tkgs-cluster-name
```

4 Erstellen Sie eine Geheimschlüsseldatei mit dem Namen `credentials-minio`. Aktualisieren Sie die Datei mit den von Ihnen erfassten MinIO-Serveranmeldedaten. Beispiel:

```
aws_access_key_id = 0XXNO8JCCGV41QZBV0RQ
aws_secret_access_key = clZ1bf8Ljkvkmq7fHucrKCkxV39BRbcycGeXQDfx
```

Hinweis Wenn Sie eine Fehlermeldung vom Typ „Fehler beim Abrufen eines Sicherungsspeichers“ mit der Beschreibung „NoCredentialProviders: keine gültigen Anbieter in Kette“ erhalten, platzieren Sie die Zeile `[default]` am Anfang der Anmeldedatendatei. Beispiel:

```
[default]
aws_access_key_id = 0XXNO8JCCGV41QZBV0RQ
aws_secret_access_key = clZ1bf8Ljkvkmq7fHucrKCkxV39BRbcycGeXQDfx
```

5 Speichern Sie die Datei und stellen Sie sicher, dass die Datei vorhanden ist.

```
ls
```

6 Führen Sie den folgenden Befehl aus, um Velero und Restic auf dem Kubernetes-Zielcluster zu installieren. Ersetzen Sie beide URLs durch die URL Ihrer MinIO-Instanz.

```
velero install \
--provider aws \
--plugins velero/velero-plugin-for-aws:v1.0.0 \
--bucket tkgs-velero \
--secret-file ./credentials-minio \
--use-volume-snapshots=false \
--use-restic \
--backup-location-config \
region=minio,s3ForcePathStyle="true",s3Url=http://10.199.17.63:9000,publicUrl=http://
10.199.17.63:9000
```

7 Überprüfen Sie die Installation von Velero und Restic.

```
kubectl logs deployment/velero -n velero
```

8 Überprüfen Sie den `velero`-Namespace.

```
kubectl get ns
```

9 Überprüfen Sie die `velero`- und `restic`-Pods.

```
kubectl get all -n velero
```

Fehlerbehebung bei Restic-DaemonSet (falls erforderlich)

Um das Restic-DaemonSet mit drei Pods auf einem Kubernetes-Cluster auszuführen, müssen Sie möglicherweise die Restic-DaemonSet-Spezifikation aktualisieren und den hostPath ändern. Weitere Informationen zu diesem Problem finden Sie unter Restic-Integration in der Velero-Dokumentation.

1 Überprüfen Sie das Restic-DaemonSet mit drei Pods.

```
kubectl get pod -n velero
```

Wenn sich die Pods im Status „CrashLoopBackOff“ befinden, bearbeiten Sie sie wie folgt.

2 Führen Sie den Befehl `edit` aus.

```
kubectl edit daemonset restic -n velero
```

3 Ändern Sie den hostPath von `/var/lib/kubelet/pods` zu `/var/vcap/data/kubelet/pods`.

```
- hostPath:
    path: /var/vcap/data/kubelet/pods
```

4 Speichern Sie die Datei.

5 Überprüfen Sie das Restic-DaemonSet mit drei Pods.

```
kubectl get pod -n velero
```

```
NAME                      READY   STATUS    RESTARTS   AGE
restic-5jln8              1/1     Running   0          73s
restic-bpvtq              1/1     Running   0          73s
restic-vg8j7              1/1     Running   0          73s
velero-72c84322d9-le7bd   1/1     Running   0          10m
```

Anpassen der Velero-Speichergrenzwerte (bei Bedarf)

Wenn Ihre Velero-Sicherung für viele Stunden den Status `status=InProgress` zurückgibt, sollten Sie die Grenzwerte und die Speichereinstellungen für Anforderungen erhöhen.

1 Führen Sie den folgenden Befehl aus.

```
kubectl edit deployment/velero -n velero
```

2 Ändern Sie die Grenzwerte und die Speichereinstellungen für Anforderungen von der Standardeinstellung von `256Mi` bzw. `128Mi` zu `512Mi` bzw. `256Mi`.

```
ports:
- containerPort: 8085
  name: metrics
  protocol: TCP
```

```
resources:
  limits:
    cpu: "1"
    memory: 512Mi
  requests:
    cpu: 500m
    memory: 256Mi
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
```

Sichern und Wiederherstellen von Clusterarbeitslasten mit eigenständigem Velero und Restic

Sie können Arbeitslasten, die in TKG-Clustern ausgeführt werden, mithilfe von eigenständigem Velero und Restic sichern und wiederherstellen. Dieses Verfahren ist eine Alternative zur Verwendung des eingebetteten Velero-Plug-In für vSphere. Eigenständiges Velero wird hauptsächlich verwendet, wenn Sie Portabilität benötigen. Restic ist für statusbehaftete Arbeitslasten erforderlich.

Voraussetzungen

Zum Sichern und Wiederherstellen von Arbeitslasten in einem TKG-Cluster mit eigenständigem Velero und Restic müssen Sie die eigenständige Version von Velero und Restic im Zielcluster installieren. Wenn die Wiederherstellung in einem separaten Zielcluster durchgeführt werden soll, müssen Velero und Restic ebenfalls im Zielcluster installiert sein. Weitere Informationen finden Sie unter Installieren und Konfigurieren von eigenständigem Velero und Restic in TKG-Clustern.

Sichern einer statusfreien Anwendung, die auf einem TKG-Cluster ausgeführt wird

Zum Sichern einer statusfreien Anwendung, die auf einem TKG-Cluster ausgeführt wird, müssen Sie Velero verwenden.

Dieses Beispiel zeigt, wie Sie eine statusfreie Beispielanwendung mit dem `--include namespaces`-Tag sichern und wiederherstellen, wobei sich alle Anwendungskomponenten in diesem Namespace befinden.

```
velero backup create example-backup --include-namespaces example-backup
```

Sie sollten folgende Meldung sehen:

```
Backup request "example-backup" submitted successfully.
Run `velero backup describe example-backup` or `velero backup logs example-backup` for more
details.
```

Überprüfen Sie die erstellte Sicherung.

```
velero backup get
```

```
velero backup describe example-backup
```

Überprüfen Sie den Velero-Bucket im S3-kompatiblen Objektspeicher, z. B. auf dem MinIO-Server.

Velero schreibt einige Metadaten in benutzerdefinierte Kubernetes-Ressourcendefinitionen (Custom Resource Definitions, CRDs).

```
kubectl get crd
```

Mit den Velero-CRDs können Sie bestimmte Befehle ausführen, wie z. B.:

```
kubectl get backups.velero.io -n velero
```

```
kubectl describe backups.velero.io guestbook-backup -n velero
```

Wiederherstellen einer statusfreien Anwendung, die auf einem TKG-Cluster ausgeführt wird

Zum Wiederherstellen einer statusfreien Anwendung, die auf einem TKG-Cluster ausgeführt wird, müssen Sie Velero verwenden.

Um die Wiederherstellung einer Beispielanwendung zu testen, löschen Sie sie.

Löschen Sie den Namespace:

```
kubectl delete ns guestbook
namespace "guestbook" deleted
```

Stellen Sie die App wieder her:

```
velero restore create --from-backup example-backup
```

Sie sollten folgende Meldung sehen:

```
Restore request "example-backup-20200721145620" submitted successfully.
Run `velero restore describe example-backup-20200721145620` or `velero restore logs example-
backup-20200721145620` for more details.
```

Überprüfen Sie, ob die App wiederhergestellt wurde:

```
velero restore describe example-backup-20200721145620
```

Führen Sie zur Überprüfung die folgenden Befehle aus:

```
velero restore get
```

```
kubectl get ns
```

```
kubectl get pod -n example
```

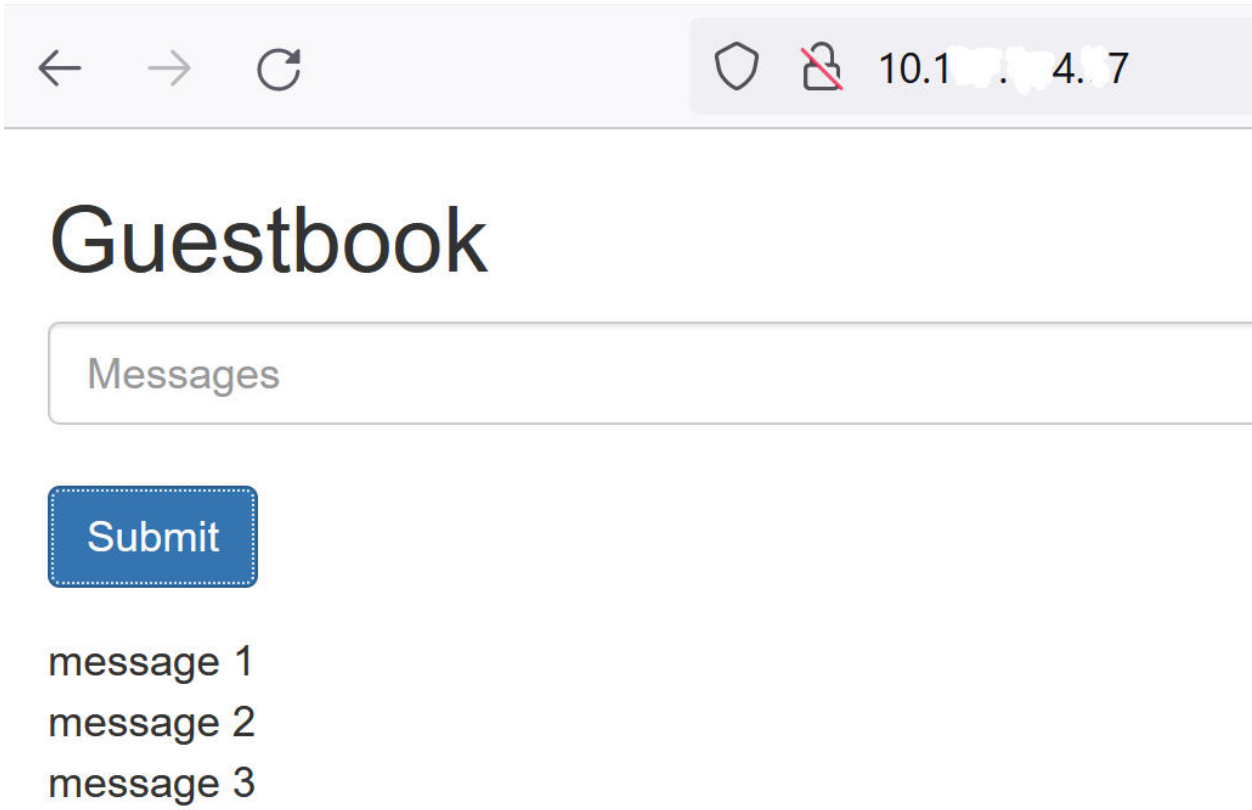
```
kubectl get svc -n example
```

Sichern einer statusbehafteten Anwendung, die auf einem TKG-Cluster ausgeführt wird

Zum Sichern einer statusbehafteten Anwendung, die auf einem TKG-Cluster ausgeführt wird, müssen sowohl die Metadaten als auch die Daten der Anwendung gesichert werden, die auf einem persistenten Volume gespeichert sind. Dazu benötigen Sie Velero und Restic.

Für dieses Beispiel verwenden wir die Guestbook-Anwendung. Dabei wird davon ausgegangen, dass Sie die Guestbook-Anwendung in einem TKG-Cluster bereitgestellt haben. Weitere Informationen finden Sie unter Bereitstellen der Guestbook-Anwendung in einem TKG-Cluster.

Zur Veranschaulichung der statusbehafteten Sicherung und Wiederherstellung senden Sie irgendeine Nachricht über die Frontend-Seite an die Guestbook-Anwendung, damit die Nachrichten dauerhaft gespeichert werden. Beispiel:



Dieses Beispiel zeigt, wie Sie die Guestbook-App mit dem `--include namespace`-Tag und Pod-Anmerkungen sichern und wiederherstellen.

Hinweis In diesem Beispiel werden Anmerkungen verwendet. Für Velero Version 1.5 und höher sind jedoch keine Anmerkungen mehr erforderlich. Um keine Anmerkungen zu verwenden, können Sie beim Erstellen der Sicherung die Option `--default-volumes-to-restic` verwenden. Dadurch werden automatisch alle PVs mit Restic gesichert. Weitere Informationen hierzu finden Sie unter https://velero.io/docs/v1.5/restic/.

Um den Sicherungsvorgang zu starten, rufen Sie die Namen der Pods ab:

```
kubectl get pod -n guestbook
```

Beispiel:

```
kubectl get pod -n guestbook

NAME                                          READY   STATUS    RESTARTS   AGE
guestbook-frontend-deployment-85595f5bf9-h8cff  1/1     Running            0          55m
guestbook-frontend-deployment-85595f5bf9-lw6tg  1/1     Running            0          55m
guestbook-frontend-deployment-85595f5bf9-wpqc8  1/1     Running            0          55m
redis-leader-deployment-64fb8775bf-kbs6s        1/1     Running            0          55m
redis-follower-deployment-84cd76b975-jrn8v      1/1     Running            0          55m
redis-follower-deployment-69df9b5688-zml4f      1/1     Running            0          55m
```

Die persistenten Volumes werden an die Redis-Pods angehängt. Da wir diese statusbehafteten Pods mit Restic sichern, müssen wir den statusbehafteten Pods mit dem Namen `volumeMount` Anmerkungen hinzufügen.

Sie müssen den `volumeMount` kennen, um den statusbehafteten Pod mit Anmerkungen zu versehen. Führen Sie folgenden Befehl aus, um den `mountName` abzurufen.

```
kubectl describe pod redis-leader-deployment-64fb8775bf-kbs6s -n guestbook
```

In den Ergebnissen sehen Sie `Containers.leader.Mounts: /data` aus `redis-leader-data`. Dieses letzte Token ist der `volumeMount`-Name, der für die Annotation des Leader-Pods verwendet werden soll. Für den Follower ist dies `redis-follower-data`. Sie können den `volumeMount`-Namen auch aus der Quell-YAML abrufen.

Versehen Sie jeden Redis-Pod mit Anmerkungen, z. B.:

```
kubectl -n guestbook annotate pod redis-leader-64fb8775bf-kbs6s backup.velero.io/backup-
volumes=redis-leader-data
```

Sie sollten folgende Meldung sehen:

```
pod/redis-leader-64fb8775bf-kbs6s annotated
```

Überprüfen Sie die Anmerkungen:

```
kubectl -n guestbook describe pod redis-leader-64fb8775bf-kbs6s | grep Annotations
Annotations:  backup.velero.io/backup-volumes: redis-leader-data
```

```
kubectl -n guestbook describe pod redis-follower-779b6d8f79-5dphr | grep Annotations
Annotations:  backup.velero.io/backup-volumes: redis-follower-data
```

Führen Sie die Velero-Sicherung durch:

```
velero backup create guestbook-backup --include-namespaces guestbook
```

Sie sollten folgende Meldung sehen:

```
Backup request "guestbook-backup" submitted successfully.
Run `velero backup describe guestbook-pv-backup` or `velero backup logs guestbook-pv-backup`
for more details.
```

Überprüfen Sie die erstellte Sicherung.

```
velero backup get

NAME                  STATUS      ERRORS   WARNINGS   CREATED
EXPIRES   STORAGE LOCATION   SELECTOR
guestbook-backup      Completed   0        0          2020-07-23 16:13:46 -0700 PDT
29d       default             <none>
```

Überprüfen Sie die Sicherungsdetails.

```
velero backup describe guestbook-backup --details
```

Beachten Sie, dass Sie mit Velero andere Befehle ausführen können, wie z. B.:

```
kubectl get backups.velero.io -n velero

NAME               AGE
guestbook-backup   4m58s
```

Und:

```
kubectl describe backups.velero.io guestbook-backup -n velero
```

Wiederherstellen einer statusbehafteten Anwendung, die auf einem TKG 2.0-Cluster ausgeführt wird

Zum Wiederherstellen einer statusbehafteten Anwendung, die auf einem TKG-Cluster ausgeführt wird, müssen sowohl die Metadaten als auch die Daten der Anwendung wiederhergestellt werden, die auf einem persistenten Volume gespeichert sind. Dazu benötigen Sie Velero und Restic.

In diesem Beispiel wird davon ausgegangen, dass Sie die statusbehaftete Guestbook-Anwendung wie im vorherigen Abschnitt beschrieben gesichert haben.

Um die Wiederherstellung der statusbehafteten Anwendung zu testen, löschen Sie ihren Namespace:

```
kubectl delete ns guestbook
namespace "guestbook" deleted
```

Überprüfen Sie, ob die Anwendung gelöscht wurde:

```
kubectl get ns
kubectl get pvc,pv --all-namespaces
```

Verwenden Sie die folgende Befehlssyntax, um eine Anwendung von der Sicherung wiederherzustellen.

```
velero restore create --from-backup <velero-backup-name>
```

Beispiel:

```
velero restore create --from-backup guestbook-backup
```

Sinngemäß sollten Meldungen wie die folgenden angezeigt werden:

```
Restore request "guestbook-backup-20200723161841" submitted successfully.
Run `velero restore describe guestbook-backup-20200723161841` or `velero restore logs
guestbook-backup-20200723161841` for more details.
```

Überprüfen Sie, ob die statusbehaftete Guestbook-Anwendung wiederhergestellt wurde:

```
velero restore describe guestbook-backup-20200723161841

Name:         guestbook-backup-20200723161841
Namespace:    velero
Labels:       <none>
Annotations:  <none>

Phase:  Completed

Backup:  guestbook-backup

Namespaces:
  Included:  all namespaces found in the backup
  Excluded:  <none>

Resources:
  Included:        *
  Excluded:        nodes, events, events.events.k8s.io, backups.velero.io,
restores.velero.io, resticrepositories.velero.io
  Cluster-scoped:  auto

Namespace mappings:  <none>

Label selector:  <none>

Restore PVs:  auto

Restic Restores (specify --details for more information):
  Completed:  3
```

Führen Sie den folgenden zusätzlichen Befehl aus, um die Wiederherstellung zu überprüfen:

```
velero restore get

NAME                                BACKUP                STATUS      ERRORS   WARNINGS
CREATED                         SELECTOR
guestbook-backup-20200723161841     guestbook-backup      Completed   0        0
2021-08-11 16:18:41 -0700 PDT   <none>
```

Überprüfen Sie, ob der Namespace wiederhergestellt wurde:

```
kubectl get ns

NAME              STATUS   AGE
default           Active   16d
guestbook         Active   76s
...
velero            Active   2d2h
```

Überprüfen Sie, ob die Anwendung wiederhergestellt wurde:

```
vkubectl get all -n guestbook

NAME                                READY   STATUS    RESTARTS   AGE
pod/frontend-6cb7f8bd65-h2pnb       1/1     Running   0          6m27s
pod/frontend-6cb7f8bd65-kwlpr       1/1     Running   0          6m27s
pod/frontend-6cb7f8bd65-snwl4       1/1     Running   0          6m27s
pod/redis-leader-64fb8775bf-kbs6s   1/1     Running   0          6m28s
pod/redis-follower-779b6d8f79-5dphr 1/1     Running   0          6m28s
pod/redis-follower-899c7e2z65-8apnk 1/1     Running   0          6m28s

NAME                                 TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)        AGE
service/guestbook-frontend           LoadBalancer   10.10.89.59      10.19.15.99
80:31513/TCP   65s
service/redis-follower               ClusterIP      10.111.163.189   <none>
6379/TCP       65s
service/redis-leader                 ClusterIP      10.111.70.189    <none>
6379/TCP       65s

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/guestbook-frontend-deployment   3/3     3            3           65s
deployment.apps/redis-follower-deployment       1/2     2            1           65s
deployment.apps/redis-leader-deployment         1/1     1            1           65s

NAME                                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/guestbook-frontend-deployment-56fc5b6b47   3         3         3       65s
replicaset.apps/redis-follower-deployment-6fc9cf5759       2         2         1       65s
replicaset.apps/redis-leader-deployment-7d89bbdbcf         1         1         1       65s
```

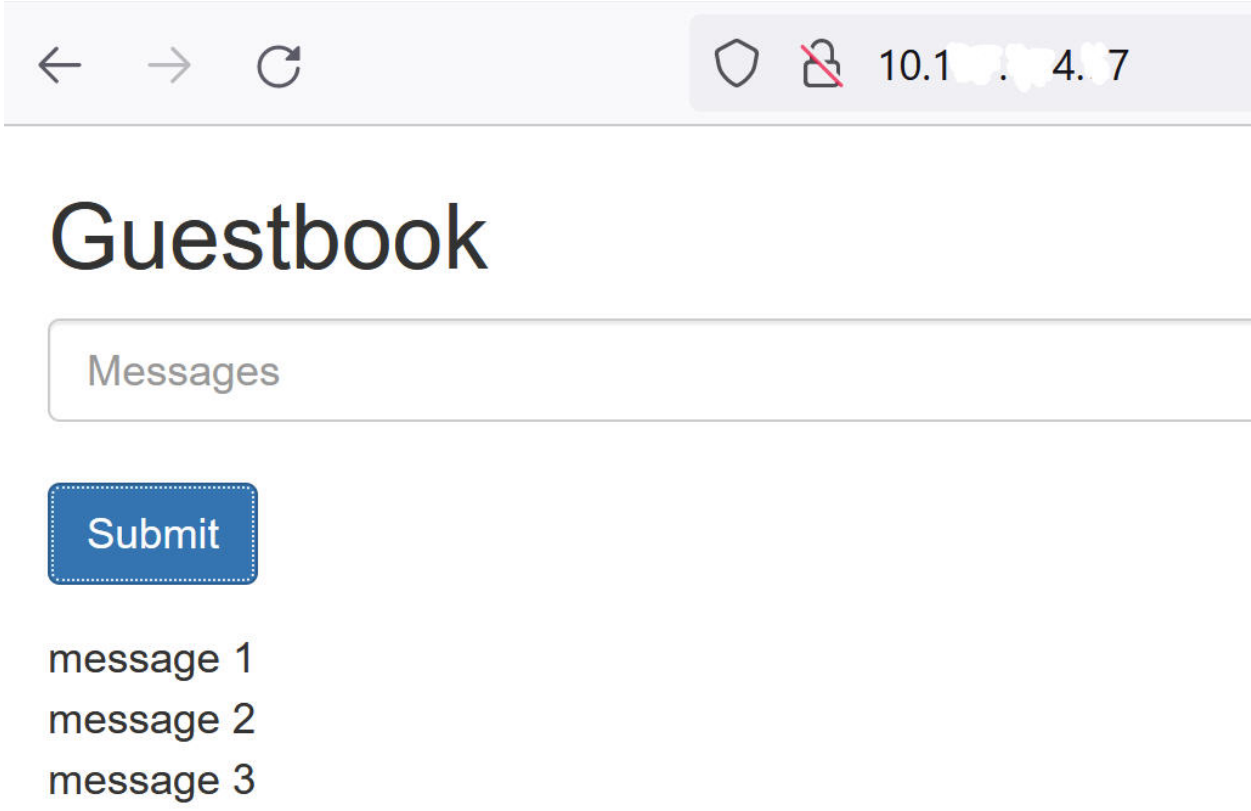
Überprüfen Sie, ob die persistenten Volumes wiederhergestellt wurden:

```
kubectl get pvc,pv -n guestbook

NAME                                        STATUS
VOLUME                                      CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/redis-leader-claim    Bound    pvc-a2f6e6d4-42db-4fb8-
a198-5379a2552509   2Gi        RWO            thin-disk      2m40s
persistentvolumeclaim/redis-follower-claim Bound    pvc-55591938-921f-452a-
b418-2cc680c0560b   2Gi        RWO            thin-disk      2m40s

NAME                                                        CAPACITY   ACCESS MODES   RECLAIM
POLICY   STATUS   CLAIM                                STORAGECLASS   REASON   AGE
persistentvolume/pvc-55591938-921f-452a-b418-2cc680c0560b   2Gi        RWO
Delete           Bound    guestbook/redis-follower-claim    thin-disk               2m40s
persistentvolume/pvc-a2f6e6d4-42db-4fb8-a198-5379a2552509   2Gi        RWO
Delete           Bound    guestbook/redis-leader-claim      thin-disk               2m40s
```

Öffnen Sie zum Schluss das Guestbook-Frontend über die externe IP des Guestbook-Frontend-Diensts und überprüfen Sie, ob die Nachrichten, die Sie zu Beginn des Lernprogramms gesendet haben, wiederhergestellt wurden. Beispiel:



Sichern und Wiederherstellen mithilfe von Velero mit CSI-Snapshot

Sie können Velero mit CSI-Snapshot verwenden, um von CSI erstellte persistente Volumes für Arbeitslasten zu sichern und wiederherzustellen, die in auf Supervisor bereitgestellten TKG-Clustern ausgeführt werden.

Anforderungen

Beachten Sie die folgenden Anforderungen:

- vSphere 8.0 U2 oder höher
- Tanzu Kubernetes-Version v1.26.5 für vSphere 8.x oder höher
- Persistente Volumes, die mithilfe von CSI-Treibern erstellt wurden, die Volume-Snapshot unterstützen

Achtung Die Verwendung von Velero mit CSI-Snapshot ist nur für dauerhafte Volumes verfügbar, die mithilfe von CSI-Treibern erstellt wurden, die Volume-Snapshots unterstützen. Weitere Informationen finden Sie unter Erstellen von Snapshots in einem TKG-Cluster in *Verwenden des TKG-Dienstes mit der vSphere IaaS-Steuerungsebene*.

Prozedur

Sie können Velero mit CSI-Snapshot (Container Storage Interface) zum Sichern und Wiederherstellen von Arbeitslasten verwenden, die auf TKGS-Clustern ausgeführt werden. Der Velero node-agent ist ein DaemonSet, das Module hostet, um die konkreten Aufgaben der Sicherung und Wiederherstellung mithilfe von CSI-Snapshot-Datenverschiebungen abzuschließen. Weitere Informationen finden Sie unter Unterstützung von Container Storage Interface Snapshots in Velero.

1 Erstellen Sie einen S3-kompatiblen Speicherort, z. B. MinIO oder einen AWS S3-Bucket.

   Im folgenden Beispiel wird ein AWS S3-Bucket verwendet.

   Informationen zur Verwendung von MinIO finden Sie unter Installieren und Konfigurieren des MinIO-Objektspeichers.

2 Installieren Sie die Velero-CLI auf dem Clusterclient, auf dem Sie kubectl ausführen.

   Laden Sie ihn von https://github.com/vmware-tanzu/velero/releases herunter.

   Weitere Informationen finden Sie in den Installationsanweisungen unter einem der folgenden Links:

   - Schritt 1: Installieren der Velero-CLI auf einem Linux-Workstation
   - Installieren der Velero-CLI
   - https://velero.io/docs/v1.12/basic-install/#install-the-cli

3 Stellen Sie eine Verbindung zu dem TKG-Dienst-Cluster her, in dem Sie die Velero-Sicherung testen möchten.

   Weitere Informationen hierzu finden Sie unter Herstellen einer Verbindung mit einem TKG-Dienst-Cluster als vCenter Single Sign-On-Benutzer mit Kubectl.

4 Führen Sie den Velero-Befehl „install“ aus, z. B. mit einem AWS S3-Speicher und der entsprechenden Anmeldedatendatei.

```
velero install \
    --provider aws \
    --plugins velero/velero-plugin-for-aws:v1.9.0,velero/velero-plugin-for-csi:v0.7.0 \
    --bucket velero-cpe-backup-bucket \
    --secret-file ./cloud-credential \
    --use-volume-snapshots=true \
    --features=EnableCSI --use-node-agent
```

Hinweis Ab Velero v1.14 wird das Velero-CSI-Plug-In mit Velero zusammengeführt. Wenn Sie demnach Velero v1.14 oder höher installieren, müssen Sie das Velero-CSI-Plug-In nicht installieren. Wenn Sie es dennoch tun, kann der Velero-Pod nicht gestartet werden.

Fehlerbehebung bei TKG-Dienstclustern

21

Sie können Fehler bei TKG-Dienstclustern mithilfe einer Vielzahl von Methoden beheben. Dazu gehören das Herstellen einer Verbindung mit einem beliebigen Clusterknoten, das Anzeigen der Cluster-Ressourcenhierarchie und das Erfassen von Protokolldateien.

Lesen Sie als Nächstes die folgenden Themen:

Abrufen von Protokollen zur Fehlerbehebung bei TKG-Clustern auf Supervisor

In diesem Abschnitt finden Sie verschiedene Protokolle für die Fehlerbehebung bei TKG-Clustern auf Supervisor, einschließlich eines Supervisor-Support-Pakets, des Arbeitslastverwaltungsprotokolls und der CAPI-, CAPV-, VM Operator- und TKG-Controller-Manager-Protokolle.

Erfassen eines Support-Pakets für Supervisor

Zur Fehlerbehebung bei TKG-Clusterfehlern können Sie die Supervisor-Protokolle exportieren. In der Regel erfolgt die Überprüfung solcher Protokolle in Absprache mit dem VMware Support.

1 Melden Sie sich mit dem vSphere Client bei Ihrer vSphere IaaS control plane-Umgebung an.

2 Wählen Sie **Menü > Arbeitslastverwaltung** aus.

3 Wählen Sie die Registerkarte **Supervisor**.

4 Wählen Sie die **Supervisor**-Zielinstanz aus.

5 Wählen Sie **Protokolle exportieren** aus.

Lesen Sie nach der Erfassung des Support-Pakets folgenden KB-Artikel: Hochladen von Diagnoseinformationen für VMware über das Secure FTP-Portal: http://kb.vmware.com/kb/2069559. Weitere Informationen finden Sie auch unter Erfassen von Protokollen für vSphere with Tanzu.

Erfassen eines Support-Pakets für ein TKG-Cluster

Sie können das Dienstprogramm TKC Support Bundler verwenden, um TKG-Clusterprotokolldateien zu erfassen und Probleme zu beheben.

Informationen zum Abrufen und Verwenden des Dienstprogramms TKC Support Bundler finden Sie im Artikel Erfassen von Protokollen für vSphere with Tanzu in der Knowledgebase des VMware-Supports.

Tailing der Protokolldatei der Arbeitslastverwaltung

Das Anpassen der Protokolldatei der Steuerungsebene für Arbeitslasten (Workload Control Plane, WCP) kann Ihnen bei der Fehlerbehebung von Supervisor- und TKG-Clusterfehlern helfen.

1 Stellen Sie eine SSH-Verbindung mit der vCenter Server Appliance her.

2 Melden Sie sich als `root`-Benutzer an.

3 Führen Sie den Befehl `shell` aus.

Es wird Folgendes angezeigt:

```
Shell access is granted to root
root@localhost [ ~ ]#
```

4 Führen Sie den folgenden Befehl aus, um das Tailing der WCP-Protokolldatei durchzuführen.

```
tail -f /var/log/vmware/wcp/wcpsvc.log
```

Erfassen der TKG-spezifischen Protokolle von Supervisor

Supervisor führt mehrere Kubernetes-Pods aus, die die Infrastruktur für TKG 2.0 bereitstellen.

```
kubectl -n vmware-system-capw get deployments.apps
NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
capi-controller-manager                       2/2     2            2           18h
capi-kubeadm-bootstrap-controller-manager     2/2     2            2           18h
capi-kubeadm-control-plane-controller-manager 2/2     2            2           18h
capv-controller-manager                       2/2     2            2           10h
capw-controller-manager                       2/2     2            2           18h
capw-webhook                                  2/2     2            2           18h
```

Bei den Infrastruktur-Pods handelt es sich um Bereitstellungen, die Replikate ausführen. Sie müssen ermitteln, welches Replikat der Leader ist, und seine Protokolle auf die neuesten Informationen überprüfen. Ein Nicht-Leader wird in der Regel aufhören, nachdem er ein Protokoll über den Versuch, die Lease abzurufen, erstellt hat.

Sie müssen sich bei Supervisor anmelden und den vSphere-Namespace von Supervisor verwenden, um diese Pods zu überprüfen.

Protokolle, die eine Bezeichnungsauswahl verwenden, funktionieren möglicherweise nicht, sodass Sie möglicherweise die zufällige Zeichenfolge ausarbeiten müssen, die am Ende des Pod-Namens hinzugefügt wird. Piping-Ausgabe an „grep 'error'“ oder „grep -i 'error'“ kann in manchen Fällen ein nützlicher Start sein. Beispielsweise `kubectl logs <args> | grep error`.

CAPI-Protokolle

Cluster-API-Anbieter:

```
kubectl logs -n vmware-system-capw -c manager vmware-system-capw-capi-controller-manager-<id>
```

CAPV-Protokolle

Cluster-API für vSphere-Anbieter:

```
kubectl logs -n vmware-system-capv -c manager vmware-system-capw-v1alpha3-vmware-system-capv-v1alpha3-controller-manager-<id>
```

VM-Operator-Protokolle

VM-Operator:

```
kubectl logs -n vmware-system-vmop -c manager vmware-system-vmop-controller-manager-<id>
```

TKG-Controller-Manager-Protokolle

GCM-Controller-Manager

```
kubectl logs -n vmware-system-tkg -c manager vmware-system-tkg-controller-manager-<id>
```

Überprüfen der Integrität der TKG-Komponenten auf Supervisor

In diesem Abschnitt finden Sie verschiedene Techniken zur Überprüfung der Integrität von Supervisor in Bezug auf TKG-Komponenten.

Überprüfen des Zustands von Supervisor-Pods

Supervisor-Pods führen TKG-Infrastrukturkomponenten aus.

Überprüfen Sie, ob sich alle Pods auf Supervisor im ausgeführten Zustand befinden.

```
kubectl get pods -A | grep "Running"
```

Hinweis Alternativ können Sie `grep -v "Running"` verwenden, um Pods zurückzugeben, die nicht ausgeführt werden.

Beispiel:

```
NAMESPACE
NAME                                                              READY   STATUS
RESTARTS        AGE
kube-system
coredns-855c5b4cfd-8w4hp                                          1/1     Running
0               27d
kube-system                                 coredns-855c5b4cfd-
bx2hk                                          1/1     Running     0               27d
kube-system                                 coredns-855c5b4cfd-
rrb5n                                          1/1     Running     0               27d
kube-system                                 docker-
registry-423f01b9b30c727e9c237a0031999b14                 1/1     Running
0               27d
kube-system                                 docker-
registry-423f568f75dcb48725b0d768b7e4bdf5                 1/1     Running
0               27d
kube-system                                 docker-
registry-423f930ca2413d96beef34526c2e61b4                 1/1     Running
0               27d
kube-system
etcd-423f01b9b30c727e9c237a0031999b14                             1/1     Running     1 (27d
ago)     27d
kube-system
etcd-423f568f75dcb48725b0d768b7e4bdf5                             1/1     Running     1 (27d
ago)     27d
kube-system
etcd-423f930ca2413d96beef34526c2e61b4                             1/1     Running     1 (27d
ago)     27d
kube-system                                 kube-
apiserver-423f01b9b30c727e9c237a0031999b14                  1/1     Running     1 (27d
ago)     27d
kube-system                                 kube-
```

```
apiserver-423f568f75dcb48725b0d768b7e4bdf5                    1/1     Running     1 (27d
ago)     27d
kube-system                                 kube-
apiserver-423f930ca2413d96beef34526c2e61b4                    1/1     Running     1 (27d
ago)     27d
kube-system                                 kube-controller-
manager-423f01b9b30c727e9c237a0031999b14          1/1     Running     0               27d
kube-system                                 kube-controller-
manager-423f568f75dcb48725b0d768b7e4bdf5          1/1     Running     0               27d
kube-system                                 kube-controller-
manager-423f930ca2413d96beef34526c2e61b4          1/1     Running     0               27d
kube-system                                 kube-
proxy-8h499                                                   1/1     Running
0               27d
kube-system                                 kube-proxy-
bm7qt                                                    1/1     Running     0               27d
kube-system                                 kube-proxy-
dnmq2                                                    1/1     Running     0               27d
kube-system                                 kube-
scheduler-423f01b9b30c727e9c237a0031999b14                    2/2     Running     13 (25d
ago)    27d
kube-system                                 kube-
scheduler-423f568f75dcb48725b0d768b7e4bdf5                    2/2     Running
0               27d
kube-system                                 kube-
scheduler-423f930ca2413d96beef34526c2e61b4                    2/2     Running
0               27d
kube-system                                 kubectl-plugin-
vsphere-423f01b9b30c727e9c237a0031999b14          1/1     Running     3 (27d ago)     27d
kube-system                                 kubectl-plugin-
vsphere-423f568f75dcb48725b0d768b7e4bdf5          1/1     Running     3 (27d ago)     27d
kube-system                                 kubectl-plugin-
vsphere-423f930ca2413d96beef34526c2e61b4          1/1     Running     3 (27d ago)     27d
kube-system                                 wcp-
authproxy-423f01b9b30c727e9c237a0031999b14                     1/1     Running
0               27d
kube-system                                 wcp-
authproxy-423f568f75dcb48725b0d768b7e4bdf5                     1/1     Running
0               27d
kube-system                                 wcp-
authproxy-423f930ca2413d96beef34526c2e61b4                     1/1     Running
0               27d
kube-system                                 wcp-
fip-423f01b9b30c727e9c237a0031999b14                           1/1     Running
0               27d
kube-system                                 wcp-
fip-423f568f75dcb48725b0d768b7e4bdf5                           1/1     Running
0               27d
kube-system                                 wcp-
fip-423f930ca2413d96beef34526c2e61b4                           1/1     Running
0               27d
svc-tmc-c63                                 agent-updater-69f6598bcd-
zrkwq                                  1/1     Running     0               27d
svc-tmc-c63                                 agentupdater-workload-27696934--1-
vz5sg                             0/1     Completed   0               35s
```

```
svc-tmc-c63                                    cluster-health-
extension-68948f657-4gpcd                             1/1     Running     0               27d
svc-tmc-c63                                    extension-manager-f8886bfb7-
vdsm9                                    1/1     Running     0               27d
svc-tmc-c63                                    extension-updater-79b4787cf6-
bwssn                                    1/1     Running     0               27d
svc-tmc-c63                                    intent-agent-66576db5bd-
lj2gk                                      1/1     Running     0               5d6h
svc-tmc-c63                                    sync-agent-
f9c68cc58-6zddj                                       1/1     Running     0               6d
svc-tmc-c63                                    tmc-agent-installer-27696934--1-
jgwvw                                0/1     Completed   0               35s
svc-tmc-c63                                    tmc-auto-attach-6488b9cd8b-
xdfzz                                    1/1     Running     0               18h
svc-tmc-c63                                    vsphere-resource-
retriever-58985c99cb-68h6v                           1/1     Running     0               18h
vmware-system-appplatform-operator-system   vmware-system-appplatform-operator-
mgr-0                            1/1     Running     0               27d
vmware-system-appplatform-operator-system   vmware-system-psp-operator-mgr-587f66646d-
xxvmr                    1/1     Running     0               27d
vmware-system-capw                             capi-controller-
manager-766c6fc449-4qqvf                             2/2     Running     423 (26d ago)   27d
vmware-system-capw                             capi-controller-manager-766c6fc449-
bcpdq                            2/2     Running     410 (26d ago)   27d
vmware-system-capw                             capi-controller-manager-766c6fc449-
rnznx                            2/2     Running     0               26d
vmware-system-capw                             capi-kubeadm-bootstrap-controller-
manager-58fd767b49-585f2         2/2     Running     402 (25d ago)   27d
vmware-system-capw                             capi-kubeadm-bootstrap-controller-
manager-58fd767b49-96q6m         2/2     Running     398 (25d ago)   27d
vmware-system-capw                             capi-kubeadm-bootstrap-controller-
manager-58fd767b49-nssgq         2/2     Running     407 (25d ago)   27d
vmware-system-capw                             capi-kubeadm-control-plane-controller-
manager-559df997b-762jr     2/2     Running     193 (26d ago)   27d
vmware-system-capw                             capi-kubeadm-control-plane-controller-
manager-559df997b-bb42s     2/2     Running     189 (26d ago)   27d
vmware-system-capw                             capi-kubeadm-control-plane-controller-
manager-559df997b-wxhqv     2/2     Running     199 (26d ago)   27d
vmware-system-capw                             capw-controller-
manager-6dd47d75b-6ncxk                              2/2     Running     400 (25d ago)   27d
vmware-system-capw                             capw-controller-manager-6dd47d75b-
k2ph4                            2/2     Running     399 (25d ago)   27d
vmware-system-capw                             capw-controller-manager-6dd47d75b-
np9sg                            2/2     Running     403 (25d ago)   27d
vmware-system-capw                             capw-
webhook-5484757c7-2pkbt                                          2/2     Running
0               27d
vmware-system-capw                             capw-webhook-5484757c7-
fkt7z                                    2/2     Running     0               27d
vmware-system-capw                             capw-webhook-5484757c7-
r85kw                                    2/2     Running     0               27d
vmware-system-cert-manager                     cert-manager-6ccbcfcd57-
lppgn                                    1/1     Running     1 (27d ago)     27d
vmware-system-cert-manager                     cert-manager-
cainjector-796f7b74db-5qvgn                            1/1     Running     3 (27d ago)     27d
```

```
vmware-system-cert-manager                  cert-manager-webhook-586948846f-
b584m                              1/1     Running     0               27d
vmware-system-csi                           vsphere-csi-
controller-6d8cfd75cd-66zbj                          6/6     Running     0               27d
vmware-system-csi                           vsphere-csi-controller-6d8cfd75cd-
b4nhz                              6/6     Running     1 (27d ago)     27d
vmware-system-csi                           vsphere-csi-controller-6d8cfd75cd-
v6hlf                              6/6     Running     0               27d
vmware-system-kubeimage                     image-controller-ff79fb5fc-
kd6ts                                    1/1     Running     0               27d
vmware-system-license-operator              vmware-system-license-operator-controller-
manager-7d555768bnxjb   1/1     Running     0               25d
vmware-system-license-operator              vmware-system-license-operator-controller-
manager-7d555768j2sb8   1/1     Running     0               25d
vmware-system-license-operator              vmware-system-license-operator-controller-
manager-7d555768w7v77   1/1     Running     0               25d
vmware-system-logging                       fluentbit-
p24gk                                               1/1     Running     0
27d
vmware-system-logging                       fluentbit-
rj2t8                                               1/1     Running     0
27d
vmware-system-logging                       fluentbit-
xx2lk                                               1/1     Running     0
27d
vmware-system-nsop                          vmware-system-nsop-controller-
manager-65b8445959-66msw            1/1     Running     0               27d
vmware-system-nsop                          vmware-system-nsop-controller-manager-65b8445959-
nm6xh            1/1     Running     0               27d
vmware-system-nsop                          vmware-system-nsop-controller-manager-65b8445959-
sv5w7            1/1     Running     0               27d
vmware-system-nsx                           nsx-ncp-6f989c9c67-
vb4x6                                          1/1     Running     5 (27d ago)     27d
vmware-system-registry                      vmware-registry-controller-
manager-7f49485b9-72kh7                2/2     Running     0               27d
vmware-system-tkg                           masterproxy-tkgs-
plugin-8npzx                                    1/1     Running     0               27d
vmware-system-tkg                           masterproxy-tkgs-plugin-
bjtsz                                       1/1     Running     0               27d
vmware-system-tkg                           masterproxy-tkgs-plugin-
v92gt                                       1/1     Running     0               27d
vmware-system-tkg                           tkgs-plugin-server-5fc4c985c7-
bz8jh                              1/1     Running     0               27d
vmware-system-tkg                           tkgs-plugin-server-5fc4c985c7-
r9wj5                              1/1     Running     0               27d
vmware-system-tkg                           tkgs-plugin-server-5fc4c985c7-
sdr55                              1/1     Running     0               27d
vmware-system-tkg                           vmware-system-tkg-controller-manager-7ffcc55df5-
dqkkm            2/2     Running     0               25d
vmware-system-tkg                           vmware-system-tkg-controller-manager-7ffcc55df5-
hkvx9            2/2     Running     0               25d
vmware-system-tkg                           vmware-system-tkg-controller-manager-7ffcc55df5-
txxrf            2/2     Running     0               25d
vmware-system-tkg                           vmware-system-tkg-state-
metrics-5bbb6d668c-7c5vt                2/2     Running     238 (26d ago)   27d
```

```
vmware-system-tkg                          vmware-system-tkg-state-metrics-5bbb6d668c-
c87zs                  2/2     Running     237 (26d ago)   27d
vmware-system-tkg                          vmware-system-tkg-state-metrics-5bbb6d668c-
wc46p                  2/2     Running     237 (26d ago)   27d
vmware-system-tkg                          vmware-system-tkg-
webhook-567f9fd68c-425xs                       2/2     Running     0               25d
vmware-system-tkg                          vmware-system-tkg-
webhook-567f9fd68c-97d6z                       2/2     Running     0               25d
vmware-system-tkg                          vmware-system-tkg-webhook-567f9fd68c-
dnkgt                        2/2     Running     0               25d
vmware-system-ucs                          upgrade-compatibility-service-5745846d58-
tpk67                    1/1     Running     0               27d
vmware-system-ucs                          upgrade-compatibility-service-5745846d58-
twxkt                    1/1     Running     0               27d
vmware-system-ucs                          upgrade-compatibility-service-5745846d58-
wzl8x                    1/1     Running     0               27d
vmware-system-vmop                         vmware-system-vmop-controller-manager-
c8499b9df-5h6f9             2/2     Running     0               27d
vmware-system-vmop                         vmware-system-vmop-controller-manager-
c8499b9df-6wgr7             2/2     Running     0               27d
vmware-system-vmop                         vmware-system-vmop-controller-manager-c8499b9df-
tvbg6             2/2     Running     0              27d
vmware-system-vmop                         vmware-system-vmop-hostvalidator-8498cc5f4d-
vqhnk                1/1     Running     0               27d
```

Wenn sich ein Pod auf Supervisor nicht im ausgeführten Zustand befindet, überprüfen Sie diesen Pod mit dem folgenden Befehl.

```
kubectl describe pod <POD Name> -n <Namespace>
```

Überprüfen des Zustands von Supervisor-Ressourcen

TKG-Controller-Ressourcen:

```
kubectl get tkc
```

Cluster-API-Ressourcen (CAPI, CABPK, CAPW, CAPV):

```
kubectl get cluster-api
```

VM-Operator-Ressourcen:

```
kubectl get virtualmachines,virtualmachineservices,virtualmachinesetresourcepolicies
```

VM-Operator-Ressource, clusterbezogen und mit der Inhaltsbibliothek synchronisiert:

```
kubectl get virtualmachineimages
```

Speicherressourcen:

```
kubectl get persistentvolumeclaims,cnsnodevmattachment,cnsvolumemetadatas
```

Netzwerkressourcen (NSX-spezifisch):

```
kubectl get service,lb,lbm,vnet,vnetif,nsxerrors,nsxnetworkinterfaces
```

Alle Supervisor-Ressourcen abrufen und in eine Datei schreiben:

```
kubectl api-resources --namespaced -o name | paste -d',' -s | xargs kubectl get -n
<namespace> > resources_in_namespace.txt
```

Sicherstellen, dass alle Cluster-API-Bereitstellungen vorhanden sind

Stellen Sie sicher, dass CAPI-, CAPW- und CAPV-Bereitstellungen vorhanden sind.

```
kubectl -n vmware-system-capw get deployments.apps
NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
capi-controller-manager                     2/2     2            2           18h
capi-kubeadm-bootstrap-controller-manager   2/2     2            2           18h
capi-kubeadm-control-plane-controller-manager   2/2     2            2           18h
capv-controller-manager                     2/2     2            2           10h
capw-controller-manager                     2/2     2            2           18h
capw-webhook                                2/2     2            2           18h
```

Überprüfen der Support-Paketdateien

Der Ordner `commands/` im Erfassen eines Support-Pakets für Supervisor enthält journalctl-Protokolle, die Details zu den Vorgängen während des WCP-Startvorgangs enthalten.

`kubectl_describe_virtualmachine.txt`

`kubectl_describe_tanzukubernetescluster.txt`

`kubectl_describe_kubeadmconfig.txt`

`kubectl-describe-pod_kube-system.txt`

`kubectl-describe-pod_vmware-system-capw.txt`

`kubectl-describe-pod_vmware-system-tkg.txt`

`kubectl-describe-pod_vmware-system-ucs.txt`

`kubectl-describe-pod_vmware-system-vmop.txt`

`kubectl_describe_cluster_resource_virtualmachineimages.txt`

`docker_images.txt`

Überprüfen der Integrität eines TKG-Clusters

Überprüfen Sie, ob sich alle Clusterknoten (VMs) im Status „Bereit“ befinden.

```
kubectl get nodes -o wide
```

```
NAME                                                    STATUS   ROLES
AGE   VERSION            INTERNAL-IP   EXTERNAL-IP   OS-IMAGE                KERNEL-
VERSION   CONTAINER-RUNTIME
tkgs-cluster-13-control-plane-dpmjj                     Ready    control-plane,master
12d   v1.22.9+vmware.1   10.244.0.25   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-control-plane-nb5r6                     Ready    control-plane,master
12d   v1.22.9+vmware.1   10.244.0.18   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-control-plane-zpcgs                     Ready    control-plane,master
12d   v1.22.9+vmware.1   10.244.0.26   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a1-gq458-9d6458d6f-c7t8c    Ready    <none>
12d   v1.22.9+vmware.1   10.244.0.24   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a1-gq458-9d6458d6f-slzvn    Ready    <none>
12d   v1.22.9+vmware.1   10.244.0.19   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a1-gq458-9d6458d6f-vzrsd    Ready    <none>
12d   v1.22.9+vmware.1   10.244.0.22   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a2-tw99z-7b547b7f85-k5h4s   Ready    <none>
12d   v1.22.9+vmware.1   10.244.0.20   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a2-tw99z-7b547b7f85-lkmdx   Ready    <none>
12d   v1.22.9+vmware.1   10.244.0.21   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
tkgs-cluster-13-worker-nodepool-a2-tw99z-7b547b7f85-qwv98   Ready    <none>
12d   v1.22.9+vmware.1   10.244.0.23   <none>        VMware Photon OS/Linux
4.19.225-3.ph3   containerd://1.5.11
```

Überprüfen Sie, ob alle Pods ausgeführt werden.

```
kubectl get pods -A
```

```
NAMESPACE
NAME                                                                READY
STATUS      RESTARTS      AGE
kube-system                    antrea-
agent-58hv7                                                         2/2     Running
0             12d
kube-system                    antrea-
agent-6x897                                                         2/2     Running
0             12d
kube-system                    antrea-
agent-7d99k                                                         2/2     Running
0             12d
kube-system                    antrea-agent-
```

```
b7vdv                                                          2/2     Running
0             12d
kube-system                   antrea-agent-
dhdlg                                                          2/2     Running
0             12d
kube-system                   antrea-agent-
mj4wx                                                          2/2     Running
0             12d
kube-system                   antrea-agent-
v7vtv                                                          2/2     Running
0             12d
kube-system                   antrea-agent-
x49gz                                                          2/2     Running     1 (12d
ago)   12d
kube-system                   antrea-agent-
z2gth                                                          2/2     Running
0             12d
kube-system                   antrea-controller-bb59f5fbf-
t6cm9                                  1/1     Running     0             12d
kube-system                   antrea-resource-
init-65b586c9db-2cbxx                                  1/1     Running     0
12d
kube-system
coredns-5f64c4fff8-2gsqn                                                  1/1
Running     0             12d
kube-system                   coredns-5f64c4fff8-
hvkg9                                                  1/1     Running     0             12d
kube-system                   docker-registry-tkgs-cluster-13-control-plane-
dpmjj                         1/1     Running     0             12d
kube-system                   docker-registry-tkgs-cluster-13-control-plane-
nb5r6                         1/1     Running     0             12d
kube-system                   docker-registry-tkgs-cluster-13-control-plane-
zpcgs                         1/1     Running     0             12d
kube-system                   docker-registry-tkgs-cluster-13-worker-nodepool-a1-
gq458-9d6458d6f-c7t8c    1/1     Running     0             12d
kube-system                   docker-registry-tkgs-cluster-13-worker-nodepool-a1-
gq458-9d6458d6f-slzvn    1/1     Running     0             12d
kube-system                   docker-registry-tkgs-cluster-13-worker-nodepool-a1-
gq458-9d6458d6f-vzrsd    1/1     Running     0             12d
kube-system                   docker-registry-tkgs-cluster-13-worker-nodepool-a2-
tw99z-7b547b7f85-k5h4s   1/1     Running     0             12d
kube-system                   docker-registry-tkgs-cluster-13-worker-nodepool-a2-
tw99z-7b547b7f85-lkmdx   1/1     Running     0             12d
kube-system                   docker-registry-tkgs-cluster-13-worker-nodepool-a2-
tw99z-7b547b7f85-qwv98   1/1     Running     0             12d
kube-system                   etcd-tkgs-cluster-13-control-plane-
dpmjj                                  1/1     Running     0             12d
kube-system                   etcd-tkgs-cluster-13-control-plane-
nb5r6                                  1/1     Running     0             12d
kube-system                   etcd-tkgs-cluster-13-control-plane-
zpcgs                                  1/1     Running     0             12d
kube-system                   kube-apiserver-tkgs-cluster-13-control-plane-
dpmjj                         1/1     Running     0             12d
kube-system                   kube-apiserver-tkgs-cluster-13-control-plane-
nb5r6                         1/1     Running     0             12d
```

```
kube-system                    kube-apiserver-tkgs-cluster-13-control-plane-
zpcgs                          1/1     Running     0             12d
kube-system                    kube-controller-manager-tkgs-cluster-13-control-plane-
dpmjj                 1/1      Running     0             12d
kube-system                    kube-controller-manager-tkgs-cluster-13-control-plane-
nb5r6                 1/1      Running     1 (12d ago)   12d
kube-system                    kube-controller-manager-tkgs-cluster-13-control-plane-
zpcgs                 1/1      Running     0             12d
kube-system                    kube-
proxy-4kp57                                                      1/1     Running
0             12d
kube-system                    kube-
proxy-5q8pw                                                      1/1     Running
0             12d
kube-system                    kube-
proxy-5th6p                                                      1/1     Running
0             12d
kube-system                    kube-
proxy-8m6mx                                                      1/1     Running
0             12d
kube-system                    kube-proxy-
dn5lp                                                       1/1     Running
0             12d
kube-system                    kube-proxy-
qgmcg                                                       1/1     Running
0             12d
kube-system                    kube-proxy-
vbq27                                                       1/1     Running
0             12d
kube-system                    kube-proxy-
xhnws                                                       1/1     Running
0             12d
kube-system                    kube-proxy-
zgfvn                                                       1/1     Running
0             12d
kube-system                    kube-scheduler-tkgs-cluster-13-control-plane-
dpmjj                          1/1     Running     0             12d
kube-system                    kube-scheduler-tkgs-cluster-13-control-plane-
nb5r6                          1/1     Running     1 (12d ago)   12d
kube-system                    kube-scheduler-tkgs-cluster-13-control-plane-
zpcgs                          1/1     Running     0             12d
kube-system                    metrics-server-774bc4dc99-
qp7tb                                      1/1     Running     0             12d
vmware-system-auth             guest-cluster-auth-
svc-6m6cd                                    1/1     Running     0             12d
vmware-system-auth             guest-cluster-auth-svc-
h44xf                                        1/1     Running     0             12d
vmware-system-auth             guest-cluster-auth-svc-
l968n                                        1/1     Running     0             12d
vmware-system-cloud-provider   guest-cluster-cloud-provider-5f87d5d7d8-
rmd78                               1/1     Running     1 (12d ago)   12d
vmware-system-csi              vsphere-csi-controller-7d858778bd-
h7zhg                                  6/6     Running     4 (12d ago)   12d
vmware-system-csi              vsphere-csi-controller-7d858778bd-
rkl98                                  6/6     Running     0             12d
```

```
vmware-system-csi              vsphere-csi-controller-7d858778bd-
snmk7                                    6/6     Running     0              12d
vmware-system-csi              vsphere-csi-
node-22fnt                                                     3/3     Running     1 (12d
ago)   12d
vmware-system-csi              vsphere-csi-
node-5jtbr                                                     3/3     Running
0             12d
vmware-system-csi              vsphere-csi-
node-87lz6                                                     3/3     Running
0             12d
vmware-system-csi              vsphere-csi-node-
gp9sf                                                  3/3     Running     0
12d
vmware-system-csi              vsphere-csi-node-
k2psv                                                  3/3     Running     0
12d
vmware-system-csi              vsphere-csi-node-
mg8bw                                                  3/3     Running     0
12d
vmware-system-csi              vsphere-csi-node-
pctmv                                                  3/3     Running     0
12d
vmware-system-csi              vsphere-csi-node-
sslrl                                                  3/3     Running     1 (12d ago)
12d
vmware-system-csi              vsphere-csi-node-
zbqbq                                                  3/3     Running     0
12d
```

Rufen Sie den TKG-Clusterstatus ab und beschreiben Sie ihn.

```
kubectl get tkc <clustername>
```

```
kubectl describe tkc <clustername>
```

Überprüfen der Integrität des TKG-Controller-Managers

Überprüfen Sie den Status und die Integrität des TKG-Controller-Managers.

```
kubectl get deployments -n vmware-system-tkg vmware-system-tkg-controller-manager -o yaml
```

Überprüfen der Integrität des VM-Operators

Pods sollten ausgeführt werden.

```
kubectl get pods -n vmware-system-vmop
NAME                                                READY   STATUS    RESTARTS   AGE
vmware-system-vmop-controller-manager-c8499b9df-5h6f9   2/2     Running   0          27d
vmware-system-vmop-controller-manager-c8499b9df-6wgr7   2/2     Running   0          27d
vmware-system-vmop-controller-manager-c8499b9df-tvbg6   2/2     Running   0          27d
vmware-system-vmop-hostvalidator-8498cc5f4d-vqhnk       1/1     Running   0          27d
```

VM-Operator erstellt die `VirtualNetworkInterface` und überprüft ihren Status. Wenn eine Knoten-VM keine IP-Adresse erhält, sollte dieser Bereich zuerst überprüft werden. Wurde diese Phase bei der Erstellung der virtuellen Maschine korrekt durchlaufen?

Der VM-Operator ist auch für den Abgleich von `VirtualMachineService` und die Aktualisierung seines Status verantwortlich. Wenn auf eine Kubernetes-API des TKG-Clusters über die externe IP nicht zugegriffen werden kann, überprüfen Sie das Protokoll des VM-Operators.

Wählen Sie beispielsweise einen der VM-Operators-Pods aus, geben Sie den Namespace und den Manager-Container an. (Der Befehl `logs` ist für einen Container. Innerhalb eines Controller-Pods befindet sich ein Manager-Container, dessen Protokolle Sie überprüfen können.)

```
kubectl logs -f vmware-system-vmop-controller-manager-c8499b9df-5h6f9 -n vmware-system-vmop
manager
```

Behebung von TKG-Clusterverbindungs- und Anmeldefehlern

Verwenden Sie diesen Abschnitt, um Fehler bei TKG-Clusterverbindungen zu beheben und sie zu protokollieren.

Fehler wegen unzureichenden Berechtigungen

Wenn Sie nicht über ausreichende Berechtigungen für den vSphere-Namespace verfügen, können Sie keine Verbindung mit dem Supervisor oder einem TKG-Cluster als vCenter Single Sign-On-Benutzer herstellen.

Das vSphere-Plug-In für kubectl gibt die Fehlermeldung „`Error from server (Forbidden)`“ zurück, wenn Sie versuchen, eine Verbindung zu einem Supervisor oder einem TKG-Cluster als vCenter Single Sign-On-Benutzer herzustellen.

Sie verfügen nicht über ausreichende Rollenberechtigungen für vSphere-Namespace, oder Ihrem Benutzerkonto wurde kein Zugriff gewährt.

Wenn Sie ein DevOps-Ingenieur sind, der den Cluster betreibt, überprüfen Sie zusammen mit Ihrem vSphere-Administrator, ob Sie die **Bearbeiten**-Berechtigungen für den vSphere-Namespace erhalten haben. Wenn Sie ein Entwickler sind, der den Cluster zum Bereitstellen von Arbeitslasten verwendet, überprüfen Sie zusammen mit Ihrem Cluster-Administrator, ob Ihnen der Clusterzugriff gewährt wurde.

Kubectl-vSphere-Anmeldefehler

Wenn Sie beim Versuch, sich mit dem vSphere-Plug-In für kubectl beim Supervisor oder TKG-Cluster anzumelden, die folgende Fehlermeldung erhalten, kann dies auf einen Anmeldefehler zurückzuführen sein.

```
Failed to get available workloads, response from the server was invalid.
```

Um Anmeldefehler zu beheben, verwenden Sie `-v=10`, um eine ausführlichere Protokollausgabe zu erhalten.

```
kubectl vsphere login --server=10.110.150.56 --vsphere-username user@vsphere.local -v=10
```

Im Folgenden wird z. B. die Verwendung der ausführlichen Ausgabe gezeigt, um einen `invalid or missing credentials`-Fehler anzuzeigen.

```
DEBU[0000] User passed verbosity level: 10
DEBU[0000] Setting verbosity level: 10
DEBU[0000] Setting request timeout:
DEBU[0000] login called as: /usr/local/bin/kubectl-vsphere login --server=10.110.150.56 --
vsphere-username user@vpshere.local -v=10
DEBU[0000] Creating wcp.Client for --server=10.110.150.56.
INFO[0000] Does not appear to be a vCenter or ESXi address.
DEBU[0000] Got response:
INFO[0000] Using user@vpshere.local as username.
DEBU[0000] Env variable KUBECTL_VSPHERE_PASSWORD is present
DEBU[0000] Error while getting list of workloads: invalid or missing credentials
FATA[0000] Failed to get available workloads, response from the server was invalid.
```

SSH zu Supervisor

Zur Behebung von Anmeldefehlern kann eine Verbindung per SSH mit dem Supervisor erforderlich sein.

Warnung Wenn Sie eine SSH-Verbindung zu einem Supervisor-Steuerungsebenenknoten herstellen, verfügen Sie über die Berechtigung, den Supervisor-Cluster dauerhaft zu beschädigen. Wenn VMware Support Nachweise dafür findet, dass ein Kunde Änderungen an Supervisor Komponenten von einem Supervisor-Steuerungsebenenknoten aus vornimmt, kann VMware-Support den Supervisor-Cluster als nicht unterstützt markieren und verlangen, dass Sie die vSphere IaaS control plane-Lösung erneut bereitstellen. Verwenden Sie diese Sitzung nur, um Netzwerke zu testen, Protokolle anzuzeigen und kubectl logs/get/describe-Befehle auszuführen. Stellen Sie in dieser Sitzung nichts bereit, löschen oder bearbeiten Sie nichts aus dieser Sitzung, ohne die ausdrückliche Genehmigung eines KB- oder VMware-Supports.

Führen Sie die folgenden Schritte aus, um eine SSH-Verbindung zu einem Supervisor-Steuerungsebenenknoten herzustellen.

1 Melden Sie sich bei vCenter mit dem Root-Benutzerkonto an.

2 Geben Sie `dcli +i` ein, um die Datencenter-CLI im interaktiven Modus zu verwenden.

3 Führen Sie den Befehl `namespacemanagement software clusters list` aus, um den Status des Supervisor zurückzugeben.

4 Geben Sie `exit` ein, um die `dcli`-Shell zu beenden.

5 Geben Sie `shell` ein, um in den Bash-Shell-Modus zu wechseln.

6 Geben Sie `/usr/lib/vmware-wcp/decyptK8Pwd.py` ein, um die IP-Adresse und das Kennwort für Supervisor abzurufen.

7 Geben Sie `ssh 10.100.150.56` für SSH zu Supervisor ein, wobei Sie die Beispiel-IP-Adresse durch die IP-Adresse ersetzen, die vom vorherigen Befehl zurückgegeben wurde.

Beheben von Fehlern in der Inhaltsbibliothek

Informationen zur Behebung von Fehlern in der Tanzu Kubernetes Release-Inhaltsbibliothek finden Sie in den Tipps in diesem Thema.

Keine TKR-Ressourcen gefunden

Ein vSphere-Administrator hat eine Inhaltsbibliothek erstellt und mit unterstützten Tanzu Kubernetes Releases synchronisiert. Sie haben die Inhaltsbibliothek dem vSphere-Namespace zugewiesen, in dem Sie TKG-Cluster bereitstellen. Sie sind bei Supervisor angemeldet und haben den Kontext auf den vSphere-Namespace umgestellt.

Wenn Sie die folgenden Befehle ausführen, wird „`No resources found`“ zurückgegeben.

```
kubectl get tanzukubernetesreleases
```

```
kubectl get tkr
```

Führen Sie zur Fehlerbehebung die folgenden Befehle aus.

```
kubectl get virtualmachineimages -A
```

```
kubectl get vmimage -o wide
```

Überprüfen Sie, ob die Inhaltsbibliothek vorhanden und mit dem Namespace registriert ist.

```
kubectl get contentsources
```

```
 kubectl get contentsourcebindings -A
```

Melden Sie sich zur Behebung des Problems bei der vCenter Server-Verwaltungsschnittstelle an. Navigieren Sie zu **Dienste**, wählen Sie **Content Library Service** aus und klicken Sie auf **Neu starten**.

Wenn das Problem dadurch nicht behoben wird, müssen Sie möglicherweise die Inhaltsbibliothek aus dem Namespace entfernen. Erstellen Sie dazu eine neue Inhaltsbibliothek, fügen Sie sie zum Namespace hinzu und entfernen Sie sie.

Das Abrufen von Bibliothekselementen schlägt fehl

Wenn Sie versuchen, einen TKG-Cluster bereitzustellen, können Sie keine Elemente aus einer abonnierten Inhaltsbibliothek abrufen, die synchronisiert und mit vSphere-Namespace verknüpft ist.

Der folgende Fehler wird angezeigt:

```
Internal error occurred: get library items failed for.
```

Wenn die abonnierte Inhaltsbibliothek Speicherkapazitätsgrenzwerte erreicht, können Sie keine TKG-Cluster bereitstellen. Die Inhaltsbibliothek wird durch angehängten Speicher gestützt. Im Laufe der Zeit, wenn immer mehr Kubernetes-Versionen freigegeben und OVA-Dateien in der Bibliothek synchronisiert werden, wird der Speicher möglicherweise bis zu seiner Höchstkapazität gefüllt.

Wenn Sie TKRs automatisch synchronisieren, sollten Sie zur manuellen Synchronisierung wechseln und nur die TKR-Images lokal speichern, die Sie benötigen. Wenn Sie bereits die bedarfsgesteuerte Synchronisierung verwenden, entfernen Sie die nicht mehr benötigten Images aus der Bibliothek. Alternativ können Sie zu einer neuen Inhaltsbibliothek migrieren.

TKR wurde in der lokalen Inhaltsbibliothek nicht gefunden

Lokale Inhaltsbibliotheken können in internetbeschränkten Umgebungen verwendet werden.

Wenn Sie die lokale Inhaltsbibliothek erstellen, haben Sie die Möglichkeit, eine Sicherheitsrichtlinie auf die Bibliothek anzuwenden. Auch wenn das Tanzu Kubernetes-Version in die Bibliothek hochgeladen wurde, steht das Release in der Bibliothek für die Nutzung durch TKG-Cluster nicht zur Verfügung, wenn eine der folgenden Bedingungen erfüllt ist.

- Das OVF-Paket in der Inhaltsbibliothek ist nicht signiert.
- Das OVF-Paket ist mit einem ungültigen Zertifikat signiert.
- Das OVF-Paket ist mit einem Zertifikat signiert, das von dem vCenter Server, auf dem die lokale Inhaltsbibliothek konfiguriert ist, nicht als vertrauenswürdig eingestuft wird.

Wenn Sie die OVA- und VMDK-Dateien in die Inhaltsbibliothek hochladen, stellen Sie sicher, dass sich das Zertifikat und die Manifestdateien im Stammverzeichnis befinden, aus dem Sie die Datei hochladen.

Beheben von Fehlern bei VM-Klassen

VM-Klassen müssen mit dem vSphere-Namespace verknüpft werden, in dem Sie TKG-Cluster bereitstellen.

Fehler bei der Bindung von VM-Klassen

Wenn Sie einen TKGS-Cluster mithilfe einer oder mehrerer VM-Klassen bereitstellen, die Sie dem zielseitigen vSphere-Namespace nicht hinzugefügt haben, erhalten Sie die Fehlermeldung „`VirtualMachineClassBindingNotFound`“ (siehe Beispiel unten).

```
conditions:
  - lastTransitionTime: "2021-04-25T02:50:58Z"
    message: 1 of 2 completed
    reason: VirtualMachineClassBindingNotFound @ Machine/test-cluster
```

```
    severity: Error
    status: "False"
    type: ControlPlaneReady
  - lastTransitionTime: "2021-04-25T02:49:21Z"
    message: 0/1 Control Plane Node(s) healthy. 0/2 Worker Node(s) healthy
    reason: WaitingForNodesHealthy
    severity: Info
    status: "False"
    type: NodesHealthy
```

Um den Fehler zu beheben, konfigurieren Sie den vSphere-Namespace mit den VM-Klassen, die Sie für Ihren TKG-Dienstcluster verwenden möchten. Führen Sie den Befehl „`kubectl get virtualmachineclass` “ aus, um die VM-Klassen anzuzeigen, die mit dem vSphere-Namespace verknüpft sind.

Hinweis Der Befehl `kubectl get virtualmachineclassbindings` ist ab vSphere 8 U3 veraltet. Der zu verwendende Befehl lautet `virtualmachineclass`.

Warnung Der Befehl `kubectl get virtualmachineclasses` gibt alle VM-Klassen zurück, die sich auf dem Supervisor befinden. Da Sie jedoch nur die VM-Klassen verwenden können, die mit dem Ziel-vSphere-Namespace zur Bereitstellung eines Clusters verknüpft sind, dient der Plural des Substantivs lediglich Informationszwecken und kann bei der Bereitstellung nicht herangezogen werden.

Beheben von Fehlern bei der TKGS-Clusterbereitstellung

Wenn Sie einen TKGS-Cluster nicht bereitstellen können, überprüfen Sie diese Liste mit den häufigsten Fehlern und beheben Sie diese.

Überprüfen der Cluster-API-Protokolle

Wenn Sie keinen TKG-Cluster erstellen können, überprüfen Sie, ob CAPW/V funktioniert.

Der CAPW/V-Controller ist die infrastrukturspezifische Implementierung der Cluster-API. CAPW/V wird über Supervisor aktiviert. CAPW/V ist eine Komponente von TKG und kann den Lebenszyklus von TKG-Clustern verwalten.

CAPW/V ist für das Erstellen und Aktualisieren des VirtualNetwork verantwortlich. Nur wenn das virtuelle Netzwerk bereit ist, kann die Erstellung der Clusterknoten verschoben werden. Hat der Workflow zur Clustererstellung diese Phase bestanden?

CAPW/V ist für das Erstellen und Aktualisieren des VirtualMachineService verantwortlich. Wurde der VirtualMachineService erfolgreich erstellt? Wurde die externe IP-Adresse abgerufen? Hat der Workflow zur Clustererstellung diese Phase bestanden?

Um diese Schritte zu beantworten, überprüfen Sie das Cluster-API-Protokoll wie folgt:

```
kubectl config use-context tkg-cluster-ns
```

```
kubectl get pods -n vmware-system-capw  | grep capv-controller
```

```
kubectl logs -n vmware-system-capw -c manager capv-controller-manager-...
```

Fehler bei der Validierung der Clusterspezifikation

Gemäß der YAML-Spezifikation darf das Leerzeichen in einem Schlüsselnamen verwendet werden. Es handelt sich um eine Skalarzeichenfolge, die ein Leerzeichen enthält und keine Anführungszeichen erfordert.

Die TKGS-Validierung lässt jedoch die Verwendung des Leerzeichens in Schlüsselnamen nicht zu. In TKGS darf ein gültiger Schlüsselname nur aus alphanumerischen Zeichen, einem Bindestrich (z. B. `key-name`), einem Unterstrich (z. B. `KEY_NAME`) oder einem Punkt (z. B. `key.name`) bestehen.

Wenn Sie das Leerzeichen in einem Schlüsselnamen in der Clusterspezifikation verwenden, wird der TKGS-Cluster nicht bereitgestellt. Im Protokoll „vmware-system-tkg-controller-manager“ wird folgende Fehlermeldung angezeigt.

```
Invalid value: \"Key Name\": a valid config key must consist of alphanumeric
characters, '-', '_' or '.' (e.g. 'key.name', or 'KEY_NAME', or 'key-name', regex used
for validation is '[-._a-zA-Z0-9]+')"
```

Entfernen Sie zum Beheben des Fehlers das Leerzeichen oder ersetzen Sie es durch ein unterstütztes Zeichen.

Fehler beim Anwenden der TKG-Cluster-YAML

Wenn Sie beim Anwenden der TKG-Cluster-YAML Fehler erhalten, führen Sie die Fehlerbehebung wie folgt durch.

Clusternetzwerk ist nicht im richtigen Zustand

Informationen zum Workflow für die TKG-Cluster-Bereitstellung:

- CAPV erstellt ein VirtualNetwork-Objekt für jedes TKG-Clusternetzwerk.
- Wenn Supervisor mit NSX-Netzwerk konfiguriert ist, überwacht NCP VirtualNetwork-Objekte und erstellt einen NSX Tier-1-Router und ein NSX-Segment für jedes VirtualNetwork.
- CAPV überprüft den Status von VirtualNetwork und fährt mit dem nächsten Schritt im Workflow fort.

Der VM-Dienstcontroller sucht nach benutzerdefinierten Objekten, die von CAPV erstellt wurden, und verwendet diese Spezifikationen, um die VMs zu erstellen und zu konfigurieren, aus denen der TKG-Cluster besteht.

NSX Container Plugin (NCP) ist ein Controller, der Netzwerkressourcen überwacht, die über die Kubernetes-API zu „etcd“ hinzugefügt wurden, und die Erstellung entsprechender Objekte in NSX orchestriert.

Jeder dieser Controller wird als Kubernetes-Pods auf der Supervisor-Steuerungsebene ausgeführt. Um Netzwerkprobleme zu beheben, überprüfen Sie das CAPV-Controller-Protokoll, das VM-Dienstprotokoll und das NCP-Protokoll.

Überprüfen Sie die Containerprotokolle, wobei `name-XXXX` der eindeutige Pod-Name ist, wenn Sie folgenden Befehl ausführen:

```
kubectl get pods -A
kubectl logs pod/name-XXXXX -c pod-name -n namesapce
```

Ungültige Anzahl der Steuerungsebenenknoten

TKG-Cluster auf Supervisor unterstützt 1 oder 3 Steuerungsebenenknoten. Wenn Sie eine andere Anzahl an Replikaten eingeben, schlägt die Clusterbereitstellung fehl.

Ungültige Speicherklasse für Steuerungsebenen-/Worker-VM

Führen Sie den folgenden Befehl aus:

```
kubectl describe ns <tkg-clsuter-namesapce>
```

Stellen Sie sicher, dass dem Namespace, in dem Sie versuchen, den TKG-Cluster zu erstellen, eine Speicherklasse zugewiesen wurde. Im vSphere-Namespace muss ein Ressourcenkontingent vorhanden sein, das auf diese Speicherklasse verweist, und die Speicherklasse muss in Supervisor vorhanden sein.

Stellen Sie sicher, dass der Name mit der Speicherklasse übereinstimmt, die in Supervisor vorhanden ist. Führen Sie `kubectl get storageclasses` Supervisor als vSphere-Administrator aus. WCP kann den Namen umwandeln, wenn das Speicherprofil auf Supervisor angewendet wird (z. B. werden Bindestriche zu Unterstrichen).

Ungültige VM-Klasse

Stellen Sie sicher, dass der in der Cluster-YAML angegebene Wert mit einer der von `kubectl get virtualmachineclass` zurückgegebenen VM-Klassen übereinstimmt. Nur gebundene Klassen können von einem TKG-Cluster verwendet werden. Eine VM-Klasse ist gebunden, wenn Sie sie zum vSphere-Namespace hinzufügen.

Der Befehl `kubectl get virtualmachineclasses` gibt alle VM-Klassen auf dem Supervisor zurück, aber nur diejenigen, die gebunden sind, können verwendet werden.

TKR-Distributionen konnten nicht gefunden werden

Es wird eine Fehlermeldung ähnlich der folgenden angezeigt:

```
"Error from server (unable to find Kubernetes distributions):
admission webhook "version.mutating.tanzukubernetescluster.run.tanzu.vmware.com"
denied the request: unable to find Kubernetes distributions"
```

Dies ist wahrscheinlich auf ein Problem mit der Inhaltsbibliothek zurückzuführen.
Um die verfügbaren Funktionen aufzulisten, verwenden Sie den Befehl `kubectl get virtualmachineimages -A`. Das Ergebnis ist das, was in Ihrer Inhaltsbibliothek verfügbar ist und synchronisiert oder hochgeladen wird.

Für TKG auf Supervisor gibt es neue TKR-Namen, die mit der neuen TKR-API kompatibel sind. Sie müssen sicherstellen, dass Sie jede TKR in der Inhaltsbibliothek korrekt benennen.

Name in Inhaltsbibliothek: `photon-3-amd64-vmi-k8s-v1.23.8---vmware.2-tkg.1-zshippable`

Entsprechender Name in der TKG-Clusterspezifikation: `version: v1.23.8+vmware.2-tkg.1-zshippable`

TKG-YAML wird angewendet. Es werden keine VMs erstellt

Wenn die TKG 2.0-Cluster-YAML gültig ist und angewendet wird, die Knoten-VMs jedoch nicht erstellt werden, führen Sie eine Fehlerbehebung wie folgt durch.

Überprüfen der CAPI-/CAPV-Ressourcen

Überprüfen Sie, ob TKG die Ressourcen auf CAPI-/CAPV-Ebene erstellt hat.

- Überprüfen Sie, ob CAPV die VirtualMachine-Ressourcen erstellt hat.
- Überprüfen Sie die VM-Operator-Protokolle, um festzustellen, warum die VM nicht erstellt wurde. Beispiel: Die OVF-Bereitstellung ist möglicherweise aufgrund unzureichender Ressourcen auf dem ESX-Host fehlgeschlagen.
- Überprüfen Sie die CAPV- und VM-Operator-Protokolle.
- Überprüfen Sie die NCP-Protokolle. NCP ist für die Realisierung von VirtualNetwork, VirtualNetworkInterface und LoadBalancer für die Steuerungsebene verantwortlich. Wenn ein Fehler im Zusammenhang mit diesen Ressourcen auftritt, kann dies ein Problem darstellen.

Fehler bei VM-Diensten

Fehler bei VM-Diensten

- `kubectl get virtualmachineservices` in Ihrem Namespace ausführen
- Wurde ein VM-Dienst erstellt?
- `kubectl describe virtualmachineservices` in Ihrem Namespace ausführen
- Werden Fehler im VM-Dienst gemeldet?

Virtuelle Netzwerkfehler

Führen Sie `kubectl get virtualnetwork` in Ihrem Namespace aus.

Wird das virtuelle Netzwerk für diesen Cluster erstellt?

Führen Sie `kubectl describe virtual network` in Ihrem Namespace aus.

Wird die virtuelle Netzwerkschnittstelle für die VM erstellt?

Die Steuerungsebene des TKG-Clusters wird nicht ausgeführt

Wenn die TKG-Steuerungsebene nicht ausgeführt wird, überprüfen Sie, ob die Ressourcen bereit waren, als der Fehler aufgetreten ist. Handelt es sich um eine Steuerungsebene für den Knotenbeitritt, die nicht aktiv ist, oder um einen Initialisierungsknoten? Überprüfen Sie außerdem, ob die Anbieter-ID im Knotenobjekt nicht festgelegt ist.

Überprüfen, ob Ressourcen bereit waren, als ein Fehler aufgetreten ist

Neben der Suche nach Protokollen hilft Ihnen die Überprüfung des Status der zugehörigen Objekte ControlPlaneLoadBalancer zu verstehen, ob die Ressourcen bereit waren, als der Fehler auftrat. Weitere Informationen finden Sie unter „Fehlerbehebung beim Netzwerk“.

Handelt es sich um eine Steuerungsebene für den Knotenbeitritt, die nicht aktiv ist, oder um einen Initialisierungsknoten?

Knotenbeitritte funktionieren manchmal nicht ordnungsgemäß. Sehen Sie sich die Knotenprotokolle für eine bestimmte VM an. Im Cluster fehlen möglicherweise Worker- und Steuerungsebenenknoten, wenn der Initialisierungsknoten nicht erfolgreich ausgeführt wird.

Anbieter-ID ist im Knotenobjekt nicht festgelegt

Wenn die VM erstellt wurde, überprüfen Sie, ob sie über IP-Adressen verfügt, und prüfen Sie dann die cloud-init-Protokolle (und ob kubeadm-Befehle werden ordnungsgemäß ausgeführt werden)

Überprüfen Sie die CAPI-Controller-Protokolle, um festzustellen, ob ein Problem vorliegt. Sie können dies mit `kubectl get nodes` auf dem TKG-Cluster überprüfen und dann prüfen, ob die Anbieter-ID auf dem Knotenobjekt vorhanden ist.

TKG-Worker-Knoten werden nicht erstellt

Wenn der TKG-Cluster und die Steuerungsebenen-VMs erstellt werden, aber keine Worker oder keine anderen VM-Objekte erstellt wurden, versuchen Sie Folgendes:

```
kubectl describe cluster CLUSTER-NAME
```

Suchen Sie nach Ressourcen für virtuelle Maschinen im Namespace. Wurden weitere erstellt?

Überprüfen Sie die CAPV-Protokolle, um festzustellen, warum keine Bootstrap-Daten der anderen virtuellen Maschinenobjekte erstellt werden.

Wenn CAPI nicht über den Lastausgleichsdienst mit der Steuerungsebene des TKG-Clusters – entweder NSX mit der IP-Adresse der Knoten-VM oder VDS mit externem Lastausgleichsdienst – kommunizieren kann, rufen Sie die kubeconfig-Datei des TKG-Clusters mithilfe des geheimen Schlüssels im Namespace ab:

Rufen Sie die kubeconfig-Datei des TKG-Clusters mithilfe des geheimen Schlüssels im Namespace ab:

```
kubectl get secret -n <namespace> <tkg-cluster-name>-kubeconfig -o jsonpath='{.data.value}' |
base64 -d
> tkg-cluster-kubeconfig; kubectl --kubeconfig tkg-cluster-kubeconfig get pods -A
```

Wenn dieser Vorgang mit der Meldung „Verbindung abgelehnt“ fehlschlägt, wurde Ihre Steuerungsebene wahrscheinlich nicht ordnungsgemäß initialisiert. Wenn eine E/A-Zeitüberschreitung vorliegt, überprüfen Sie die Konnektivität mit der IP-Adresse in „kubeconfig“.

NSX mit eingebettetem Lastausgleichsdienst:

- Stellen Sie sicher, dass der Lastausgleichsdienst für die Steuerungsebene aktiv und erreichbar ist.
- Wenn der Lastausgleichsdienst nicht über eine IP verfügt, überprüfen Sie die NCP-Protokolle und die NSX-T Benutzeroberfläche, um festzustellen, ob sich die zugehörigen Komponenten in den richtigen Zuständen befinden. (NSX-T LB, VirtualServer, ServerPool sollten sich alle in einem fehlerfreien Zustand befinden.)
- Wenn LB eine IP hat, aber nicht erreichbar ist (`curl -k https://<LB- VIP>:6443/healthz` sollte den Fehler „nicht autorisiert“ zurückgeben).

  Wenn sich der LoadBalancer-Typ der externen IP-Adresse des Diensts im Status „ausstehend“ befindet, überprüfen Sie, ob der TKG-Cluster über die LB-VIP des Supervisors mit der Kubernetes-API des Supervisors kommunizieren kann. Stellen Sie sicher, dass sich keine IP-Adressen überschneiden.

Überprüfen, ob sich TKG-Steuerungsebenenknoten in einem fehlerfreien Zustand befinden:

- Überprüfen Sie, ob die Steuerungsebene des TKG-Clusters einen Fehler meldet (wie z. B. „Knoten mit Anbieter-ID kann nicht erstellt werden“).
- Der Cloud-Anbieter des TKG-Clusters hat den Knoten nicht mit der korrekten Anbieter-ID markiert, daher kann CAPI die Anbieter-ID im Gastclusterknoten und die Maschinenressource im Supervisor-Cluster nicht vergleichen, um sie zu überprüfen.

Melden Sie sich per SSH bei der Steuerungsebenen-VM an oder verwenden Sie die kubeconfig-Datei des TKG-Clusters, um zu überprüfen, ob der Cloud Provider Pod von TKG ausgeführt wird bzw. Fehler protokolliert wurden. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu TKG-Dienst-Clustern als Kubernetes-Administrator und Systembenutzer.

```
kubectl get po -n vmware-system-cloud-provider
```

```
kubectl logs -n vmware-system-cloud-provider <pod name>
```

Wenn der VirtualMachineService-Abgleich durch VMOP nicht erfolgreich war, überprüfen Sie das VM-Operator-Protokoll.

Wenn NCP Probleme beim Erstellen von NSX-T-Ressourcen hatte, überprüfen Sie das NCP-Protokoll.

Wenn die Steuerungsebene nicht ordnungsgemäß initialisiert wurde, ermitteln Sie die VM-IP. Der Status sollte die VM-IP enthalten.

```
kubectl get virtualmachine -n <namespace> <TKC-name>-control-plane-0 -o yaml
```

```
ssh vmware-system-user@<vm-ip> -i tkc-cluster-ssh
```

Überprüfen Sie, ob „kubeadm“ Fehler protokolliert hat.

```
cat /var/log/cloud-init-output.log | less
```

Bereitgestellter TKG-Cluster bleibt in der Phase „Wird erstellt“ hängen

Führen Sie die folgenden Befehle aus, um den Status des Clusters zu überprüfen.

```
kubectl get tkc -n <namespace>
```

```
kubectl get cluster -n <namespace>
```

```
kubectl get machines -n <namespace>
```

KubeadmConfig war vorhanden, konnte von CAPI aber nicht gefunden werden. Es wurde überprüft, ob das Token in „vmware-system-capv“ über die erforderlichen Berechtigungen zum Abfragen von „kubeadmconfig“ verfügte.

```
$kubectl --token=__TOKEN__ auth can-i get kubeadmconfig
yes
```

Es ist möglich, dass der Controller-Laufzeit-Cache nicht aktualisiert wurde. Die CAPI-Watch-Caches sind möglicherweise veraltet und nehmen die neuen Objekte nicht auf. Starten Sie bei Bedarf „capi-controller-manager“ neu, um das Problem zu beheben.

```
kubectl rollout restart deployment capi-controller-manager -n vmware-system-capv
```

vSphere-Namespace bleibt in der Phase „Wird beendet“ hängen

Stellen Sie sicher, dass TKR, Supervisor und vCenter im Hinblick auf die Versionskompatibilität synchron sind.

Namespaces können nur gelöscht werden, wenn alle Ressourcen unter den Namespaces wiederum gelöscht werden.

```
kubectl describe namespace NAME
```

Der folgende Fehler wurde gefunden: „Error from server
(unable to find Kubernetes distributions): admission webhook
"version.mutating.tanzukubernetescluster.run.tanzu.vmware.com" denied the request: unable to find Kubernetes distributions“

Überprüfen Sie die Images der virtuellen Maschine in vCenter.

```
kubectl get virtualmachineimages -A
```

Beheben von Fehlern bei TKG-Dienstclusterknoten

Wenn Sie einen TKGS-Cluster bereitgestellt haben und die Knoten erstellt werden, aber eine oder mehrere virtuelle Maschinen nicht gestartet werden oder Fehler aufweisen, beachten Sie diese Tipps zur Fehlerbehebung.

Überprüfen der relevanten CRDs

Sobald die VMs erstellt wurden, sollten die relevanten CRDs erstellt werden. Prüfen Sie, ob relevante CRDs erstellt und vorhanden sind (Maschinenbereitstellungen, virtuelle Maschinen).

Maschinenbereitstellungen prüfen:

```
kubectl get machinedeployments -A -o wide
```

Virtuelle Maschinen prüfen:

```
kubectl get virtualmachines -A
```

Knotengröße prüfen

Sie haben einen TKG-Cluster bereitgestellt. Das System versucht, die VM(s) der Steuerungsebene einzuschalten. Es wird jedoch die folgende Fehlermeldung angezeigt.

```
The host does not have sufficient CPU resources to satisfy the reservation.
```

Die Größe oder die Klasse der virtuellen Maschine reicht für die Clusterbereitstellung nicht aus. Ändern Sie den Typ oder die Klasse der virtuellen Maschine. Vermeiden Sie die Verwendung der besonders kleinen und kleinen VM-Klassentypen sowohl für die Steuerungsebene als auch für Worker-Knoten.

Beheben von Fehlern beim TKG-Dienstclusternetzwerk

Informationen zur Fehlerbehebung beim TKGS-Clusternetzwerk finden Sie in den Tipps in diesem Abschnitt.

Überprüfen des Knotennetzwerks

Jeder TKG-Cluster sollte über die folgenden Netzwerkressourcen verfügen.

Netzwerkobjekt	Netzwerkressourcen	Beschreibung	Beheben	Befehl
VirtualNetwork	Tier-1-Router und verknüpftes Segment	Knotennetzwerk für den Cluster	Stellen Sie sicher, dass die SNAT-IP zugewiesen ist	`kubectl get virtualnetwork -n NS-NAME`
VirtualNetworkInterface	Logischer Port auf Segment	Knotennetzwerkschnittstelle für Clusterknoten	Sicherstellen, dass jede virtuelle Maschine über eine IP-Adresse verfügt	`kubectl get virtualmachines -n NS-NAME NODE-NAME`

Überprüfen des Lastausgleichsdiensts für die Steuerungsebene

Der Lastausgleichsdienst für die Steuerungsebene des TKG-Clusters bietet Zugriff auf den Kubernetes-API-Server. Dieser Lastausgleichsdienst wird während der Clustererstellung automatisch vom System bereitgestellt. Es sollte über die folgenden Ressourcen verfügen.

Die Überprüfung des Status des Lastausgleichsdiensts der Steuerungsebene kann Ihnen dabei helfen, zu verstehen, ob Ressourcen bereit waren, als Fehler aufgetreten sind. Im Allgemeinen finden Sie diesen Lastausgleichsdienst unter Verwendung von „Diese Lastausgleichsdienste suchen“ mit diesem Befehl für den Supervisor-Cluster: `kubectl get services -A | grep control-plane-service`

Netzwerkobjekt	Netzwerkressourcen	Beschreibung	Beheben	Befehl
VirtualMachineService	Nicht verfügbar	VirtualMachineService wird erstellt und in einen k8s-Dienst übersetzt.	Stellen Sie sicher, dass der Status aktualisiert ist und die virtuelle IP (VIP) des Lastausgleichsdiensts enthält.	`kubectl get virtualmachineservices -n NS-NAME SERVICE-NAME`

Dienst	Lastausgleichsserver mit VirtualServer-Instanz und zugeordneter Serverpool (Mitgliederpool)	Der Kubebernetes-Dienst vom Typ „Lastausgleichsdienst “ wird für den Zugriff auf den TKG-Cluster-API-Server erstellt.	Stellen Sie sicher, dass eine externe IP-Adresse zugewiesen ist. Stellen Sie sicher, dass Sie über die externe IP des LB-Diensts auf die TKG-Cluster-API zugreifen können.	Supervisor-Namespace:
				`kubectl get services -A | grep control-plane-service`
				Cluster-Namespace:
				`kubectl get services -n NS-NAME`
				Beide Namespaces
				`curl -k https://EXTERNAL-IP:PORT/healthz`

Endpoints	Die Endpoint-Mitglieder (Knoten der Steuerungsebene des TKG-Clusters) sollten sich im Mitgliedspool befinden.	Ein Endpoint wird erstellt, um alle Steuerungsebenenknoten des TKG-Clusters einzubeziehen.		`kubectl get endpoints -n NS-NAME SERVICE-NAME`

Überprüfen der Lastausgleichsdienste auf Worker-Knoten

Eine Lastausgleichsdienst-Instanz für die Worker-Knoten des TKG-Clusters wird vom Benutzer erstellt, wenn ein Kubernetes-Dienst vom Typ „LoadBalancer“ erstellt wird.

Der erste Schritt besteht darin, sicherzustellen, dass der Cloud-Anbieter auf dem TKG-Cluster ausgeführt wird.

```
kubectl get pods -n vmware-system-cloud-provider
```

Stellen Sie sicher, dass verwandte Kubernetes-Objekte erstellt wurden und sich im richtigen Zustand befinden.

Netzwerkobjekte	Netzwerkressourcen	Beschreibung	Befehl
VirtualMachineService in Supervisor	Nicht verfügbar	Ein VirtualMachineService wird im Supervisor erstellt und in einen Kubernetes-Dienst im Supervisor übersetzt	`kubectl get virtualmachineservice -n NS-NAME SVC-NAME`
Lastausgleichsdienst im Supervisor	VirtualServer im TKG-Cluster-Lastausgleichsdienst und ein zugehöriger Mitgliederpool.	Der Lastausgleichsdienst wird im Supervisor für den Zugriff auf diesen LB-Diensttyp erstellt	`kubectl get services -n NS-NAME SVC-NAME`
Endpoints im Supervisor	Die Endpoint-Mitglieder (TKG-Cluster-Worker-Knoten) sollten sich im Mitgliedspool in NSX befinden.	Ein Endpoint wird erstellt, um alle Worker-Knoten des TKG-Clusters einzubeziehen	`# kubectl get endpoints -n NS-NAME SVC-NAME`
Lastausgleichsdienst im TKG-Cluster	Nicht verfügbar	Für den Lastausgleichsdienst im vom Benutzer bereitgestellten TKG-Cluster sollte der Status mit der Lastausgleichsdienst-IP aktualisiert werden	`kubectl get services`

Prüfen des Supervisor-NSX-Netzwerk-Stacks

Der Kubernetes-API-Server, der NCP-Pod und der Manager-Container, der in einem beliebigen Controller-Pod ausgeführt wird, sind die primären Ausgangspunkte für die Überprüfung von Infrastruktur-Netzwerkproblemen.

Die folgende Fehlermeldung kann auf ein Routing- oder MTU-Problem im Netzwerk-Fabric hindeuten, einschließlich der physischen Portgruppe, mit der die ESXi-Host-Netzwerkkarten verbunden sind:

```
{"log":"I0126 19:40:15.347154 1 log.go:172] http: TLS handshake error from
100.64.128.1:4102: EOF\n","stream":"stderr","time":"2021-01-26T19:40:15.347256146Z"}
```

Stellen Sie zur Fehlerbehebung über SSH eine Verbindung mit dem ESXi-Host her und führen Sie den folgenden Befehl aus:

```
esxcli network ip interface ipv4 get
```

Dieser Befehl listet alle VMkernel-Schnittstellen des Hosts auf. Wenn Sie über eine einzelne TEP-Schnittstelle verfügen, handelt es sich dabei immer um vmk10. Wenn Sie über eine zweite oder dritte TEP-Schnittstelle verfügen, sind dies vmk11 und vmk12 usw. Die Anzahl der erstellten TEP-Schnittstellen hängt davon ab, wie viele Uplinks Sie dem TEP im Uplink-Profil zugewiesen haben. Pro Uplink wird eine TEP-Schnittstelle erstellt, wenn Sie für die TEPs die Lastverteilung („Load Sharing“) über Uplinks ausgewählt haben.

Der Haupt-Ping-Befehl für TEP-zu-TEP-Verbindungen weist die folgende Syntax auf:

```
vmkping ++netstack=vxlan -s 1572 -d -I vmk10 10.218.60.66
```

Dabei

- ist `-s` die Paketgröße
- bedeutet `-d`, dass keine Fragmentierung erfolgen soll
- bedeutet `-I`, dass der Link von vmk10 bezogen werden soll
- ist die `IP address` eine TEP-Schnittstelle auf einem anderen ESXi-Host oder NSX Edge, den Sie pingen

Wenn die MTU auf 1600 festgelegt ist, schlägt eine Paketgröße von über 1573 wahrscheinlich fehl (Sie benötigen nur eine MTU über 1500). Wenn die MTU auf 1500 festgelegt ist, schlagen wahrscheinlich alle Paketgrößen über 1473 fehl. Sie sollten die Schnittstelle im Befehl in vmk11 ändern, wenn Sie über zusätzliche TEP-Schnittstellen verfügen, von denen Sie den Ping-Befehl absetzen möchten.

Neustarten eines fehlgeschlagenen Upgrades des TKG-Clusters

Wenn das Update eines TKG-Clusters fehlschlägt, können Sie den Aktualisierungsauftrag neu starten und die Aktualisierung erneut versuchen.

Problem

Das Aktualisieren eines TKG-Clusters schlägt fehl, was zum Clusterstatus „`upgradefailed`“ führt.

Ursache

Es kann verschiedene Gründe für eine fehlgeschlagene Clusteraktualisierung geben, z. B. nicht genügend Speicher. Führen Sie den folgenden Vorgang aus, um einen fehlgeschlagenen Aktualisierungsauftrag neu zu starten und die Aktualisierung erneut durchzuführen.

Lösung

1. Melden Sie sich beim Supervisor als Administrator an.
2. Suchen Sie den `update_job_name`.

   ```
   kubectl get jobs -n vmware-system-tkg -l "run.tanzu.vmware.com/cluster-namespace=$
   {cluster_namespace},cluster.x-k8s.io/cluster-name=${cluster_name}"
   ```

3. Führen Sie den `kubectl proxy` aus, damit `curl` für die Ausstellung von Anforderungen verwendet werden kann.

   ```
   kubectl proxy &
   ```

Sie sollten `Starting to serve on 127.0.0.1:8001` sehen.

Hinweis Sie können `kubectl` nicht verwenden, um den `.status` einer Ressource zu patchen oder zu aktualisieren.

4 Verwenden Sie `curl` mit dem folgenden Patch, um das `.spec.backoffLimit` zu erhöhen.

```
curl -H "Accept: application/json" -H "Content-Type: application/json-patch+json"
--request PATCH --data '[{"op": "replace", "path": "/spec/backoffLimit", "value": 8}]'
http://127.0.0.1:8001/apis/batch/v1/namespaces/vmware-system-tkg/jobs/${update_job_name}
```

5 Verwenden Sie `curl` mit dem folgenden Patch-Befehl, um `.status.conditions` zu löschen, sodass die Auftragssteuerung neue Pods generieren kann.

```
$ curl -H "Accept: application/json" -H "Content-Type: application/json-patch+json"
--request PATCH --data '[{"op": "remove", "path": "/status/conditions"}]'
http://127.0.0.1:8001/apis/batch/v1/namespaces/vmware-system-tkg/jobs/${update_job_name}/
status
```

Fehlerbehebung bei der Container-Bereitstellung

Container-Bereitstellungsfehler können auftreten, wenn die Pod-Sicherheitsrichtlinie und die rollenbasierte Zugriffssteuerung nicht für authentifizierte Benutzer konfiguriert sind.

Problem

Sie stellen eine Container-Arbeitslast in einem TKG 2.0-Cluster bereit, aber die Arbeitslast wird nicht gestartet. Es wird eine Fehlermeldung ähnlich der folgenden angezeigt:

```
Error: container has runAsNonRoot and image will run as root.
```

Ursache

TKG-Cluster werden mit aktivierter PodSecurityPolicy-Zugangssteuerung bereitgestellt. Kein authentifizierter Benutzer kann Pods mit oder ohne Rechte erstellen, bis der Cluster-Administrator PodSecurityPolicy an die authentifizierten Benutzer gebunden hat.

Lösung

Erstellen Sie mithilfe von TKR 1.24 oder früher eine geeignete Bindung mit der Standard PodSecurityPolicy oder definieren Sie eine benutzerdefinierte PodSecurityPolicy-Ressource. Konfigurieren Sie PSA (Pod Security Admission) bei Verwendung von TKR 1.25 oder höher. Weitere Informationen hierzu finden Sie unter Kapitel 18 Verwalten von Sicherheit für TKG-Dienstcluster.

Fehlerbehebung bei der Containerregistrierung

Sie können eine externe Containerregistrierung mit TKG-Cluster-Pods verwenden.

Fehlerbehebung beim Abrufen von Images aus einer Containerregistrierung

Wenn Sie das TKG mit den vertrauenswürdigen Zertifikaten konfigurieren und das selbstsignierte Zertifikat zur kubeconfig-Datei des Clusters hinzufügen, sollten Sie in der Lage sein, erfolgreich ein Container-Image aus einer privaten Registrierung abzurufen, welche dieses selbstsignierte Zertifikat verwendet.

Mit dem folgenden Befehl können Sie ermitteln, ob das Container-Image für eine Pod-Arbeitslast erfolgreich abgerufen wurde:

```
kubectl describe pod PODNAME
```

Diese Befehle zeigen detaillierte Status- und Fehlermeldungen für einen bestimmten Pod an. Beispiel für den Versuch, ein Image vor dem Hinzufügen benutzerdefinierter Zertifikate zum Cluster abzurufen:

```
Events:
  Type     Reason                          Age                From               Message
  ----     ------                          ----               ----               -------
  Normal   Scheduled                       33s                default-scheduler  ...
  Normal   Image                           32s                image-controller   ...
  Normal   Image                           15s                image-controller   ...
  Normal   SuccessfulRealizeNSXResource  7s (x4 over 31s)  nsx-container-ncp  ...
  Normal   Pulling                         7s                 kubelet            Waiting test-gc-
e2e-demo-ns/testimage-8862e32f68d66f727d1baf13f7eddef5a5e64bbd-v10612
  Warning  Failed                          4s                 kubelet            failed to get
images: ... Error: ... x509: certificate signed by unknown authority
```

Und wenn Sie den folgenden Befehl ausführen:

```
kubectl get pods
```

Der `ErrImagePull` wird auch in der allgemeinen Pod-Statusansicht angezeigt:

```
NAME                                        READY   STATUS         RESTARTS   AGE
testimage-nginx-deployment-89d4fcff8-2d9pz   0/1     Pending        0          17s
testimage-nginx-deployment-89d4fcff8-7kp9d   0/1     ErrImagePull   0          79s
testimage-nginx-deployment-89d4fcff8-7mpkj   0/1     Pending        0          21s
testimage-nginx-deployment-89d4fcff8-fszth   0/1     ErrImagePull   0          50s
testimage-nginx-deployment-89d4fcff8-sjnjw   0/1     ErrImagePull   0          48s
testimage-nginx-deployment-89d4fcff8-xr5kg   0/1     ErrImagePull   0          79s
```

Die Fehler „x509: Zertifikat von unbekannter Zertifizierungsstelle signiert“ und „ErrImagePull“ geben an, dass der Cluster nicht mit dem korrekten Zertifikat konfiguriert ist, um eine Verbindung zur Registrierung des privaten Containers herzustellen. Entweder fehlt das Zertifikat oder es ist falsch konfiguriert.

Wenn nach der Konfiguration der Zertifikate Fehler beim Herstellen einer Verbindung zu einer privaten Registrierung auftreten, können Sie überprüfen, ob in der Konfiguration angewendete Zertifikate auf den Cluster angewendet werden. Sie können mithilfe von SSH überprüfen, ob die Zertifikate ordnungsgemäß in ihrer Konfiguration angewendet wurden.

Zwei Ermittlungsschritte können durchgeführt werden, indem eine Verbindung zu einem Worker-Knoten über SSH erfolgt.

1 Überprüfen Sie den Ordner `/etc/ssl/certs/` auf Dateien mit dem Namen `tkg-<cert_name>.pem`, wobei `<cert_name>` die Eigenschaft „name“ des Zertifikats ist, das in `TkgServiceConfiguration` hinzugefügt wurde. Wenn die Zertifikate mit jenen in der `TkgServiceConfiguration` übereinstimmen und die Verwendung einer privaten Registrierung weiterhin nicht funktioniert, fahren Sie mit der Diagnose fort, indem Sie den nächsten Schritt ausführen.

2 Führen Sie mithilfe von selbstsignierten Zertifikaten den folgenden openssl-Verbindungstest für den Zielserver durch, indem Sie den Befehl `openssl s_client -connect hostname:port_num` ausführen, wobei `hostname` der Hostname/DNS-Name der privaten Registrierung ist, die selbstsignierte Zertifikate verwendet, und `port_num` die Nummer des Ports ist, auf dem der Dienst ausgeführt wird (für HTTPS in der Regel 443).

   Sie können überprüfen, welcher Fehler von openssl ausgegeben wird, wenn Sie versuchen, eine Verbindung mit dem Endpunkt herzustellen, der selbstsignierte Zertifikate verwendet, und die Situation von dort aus beheben, indem Sie beispielsweise die richtigen Zertifikate zur `TkgServiceConfiguration` hinzufügen. Wenn der TKG-Cluster mit dem falschen Zertifikat eingebettet ist, müssen Sie die Konfiguration mit den richtigen Zertifikaten aktualisieren, den TKG-Cluster löschen und ihn dann mit der Konfiguration neu erstellen, die die korrekten Zertifikate enthält.

3 Stellen Sie sicher, dass der Inhalt der Datenzuordnung des geheimen Schlüssels doppelt base64-codiert ist. Doppelte Base64-Codierung ist erforderlich. Wenn der Inhalt des Datenzuordnungswerts nicht doppelt Base64-codiert ist, kann die resultierende PEM-Datei nicht verarbeitet werden.

Fehlerbehebung bei zusätzlichen vertrauenswürdigen Zertifizierungsstellen

Wenn beim Hinzufügen zusätzlicher vertrauenswürdiger CA-Zertifikate zu einem TKG-Cluster Probleme auftreten, finden Sie weitere Informationen hierzu in diesem Thema.

Fehlerbehebung bei zusätzlichen vertrauenswürdigen Zertifizierungsstellen

Mithilfe der v1alpha3- oder v1beta1-API können Sie eine `trust`-Variable einschließen, die Werte für zusätzliche vertrauenswürdige CA-Zertifikate enthält. Der allgemeine Anwendungsfall besteht im Hinzufügen eines Zertifikats für eine private Containerregistrierung zum Cluster. Weitere Informationen hierzu finden Sie unter Integrieren von TKG-Dienst-Clustern in eine private Containerregistrierung.

Beispiel: Verwendung der v1beta1-API:

```
topology:
  variables:
    - name: trust
      value:
        additionalTrustedCAs:
          - name: my-ca
```

Mit folgendem geheimem Schlüssel:

```
apiVersion: v1
data:
  my-ca: # Double Base64 encoded CA certificate
kind: Secret
metadata:
  name: CLUSTER_NAME-user-trusted-ca-secret
  namespace: tap
type: Opaque
```

Wenn Sie eine `trust.additionalTrustedCAs`-Instanz zu einer TKG-Clusterspezifikation hinzufügen, aktualisiert Supervisor die Clusterknoten als Grundlegendes zum Modell für parallele Updates für TKG-Dienstcluster. Wenn es jedoch zu einem Fehler bei den `trust`-Werten kommt, werden die Maschinen nicht ordnungsgemäß gestartet und können dem Cluster nicht beitreten.

Bei Verwendung der v1beta1-API muss der Inhalt des Zertifikats doppelt Base64-codiert sein. Wenn der Inhalt des Zertifikats nicht doppelt Base64-codiert ist, wird unter Umständen folgender Fehler angezeigt.

```
ls cannot access '/var/tmp/_var_ib_containerd': No such file or directory
```

Bei Verwendung der v1alpha3-API (oder der v1alpha2-API) muss der Inhalt des Zertifikats einzeln Base64-codiert sein. Wenn der Inhalt des Zertifikats nicht Base64-codiert ist, wird unter Umständen folgender Fehler angezeigt.

```
"default.validating.tanzukubernetescluster.run.tanzu.vmware.com" denied the request:
Invalid certificate internalharbor, Error decoding PEM block for internalharbor in the
TanzuKubernetesCluster spec's trust configuration
```

Wenn Sie nicht die richtige Kodierung verwenden, werden die Maschinenknoten nicht angezeigt, und obiger Fehler wird angezeigt. Kodieren Sie zur Behebung des Problems den Inhalt des Zertifikats und fügen Sie den Wert zur Datenzuordnung des geheimen Schlüssels hinzu.

Sie können „kubectl replace -f /tmp/kubectl-edit-2005376329.yaml“ ausführen, um dieses Update zu wiederholen.