vSphere IaaS 制御プレーンでの TKG サービスの使用

Update 3

VMware vSphere 8.0
VMware vCenter 8.0
VMware ESXi 8.0

vmware by Broadcom

VMware by Broadcom の Web サイトで最新の技術ドキュメントを確認できます

https://docs.vmware.com/jp/

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

Copyright © 2023-2024 Broadcom. All Rights Reserved. 「Broadcom」という語表現は、Broadcom Inc. およびその子会社のいずれかまたは両方を指します。詳細については、https://www.broadcom.com を参照してください。 本書に記載されるすべての商標、製品名、サービス マークおよびロゴは、各社に帰属します。

目次

vSphere IaaS 制御プレーンでの TKG サービスの使用

このドキュメントでは、vSphere IaaS control plane で スーパーバイザー の Tanzu Kubernetes Grid クラスタのライフサイクルを管理する方法について説明します。

スーパーバイザー 上の Tanzu Kubernetes Grid を使用して、さまざまなスタイルのワークロード クラスタ（適切に定義されたデフォルトを持つ Tanzu Kubernetes クラスタ、幅広い定義オプションを持つクラスタ クラスのクラスタなど）をプロビジョニングできます。vSphere Zone 全体に スーパーバイザー をデプロイすることで、フォルト トレラント ワークロードがサポートされている高可用性 TKG クラスタをプロビジョニングできます。更新された Tanzu Kubernetes リリース形式は、Tanzu パッケージ リポジトリをネイティブに統合し、複数のオペレーティング システムをサポートします。スーパーバイザー 上の TKG クラスタに対し、vCenter Single Sign-On と vSphere 向け Kubernetes CLI Tools、または外部 ID プロバイダと Tanzu CLI を使用したアクセスと管理が可能です。

対象読者

このドキュメントは、vSphere IaaS control plane で スーパーバイザー の TKG クラスタのライフサイクルをプロビジョニングし、管理する、vSphere 管理者と Kubernetes オペレータが対象です。

適用可能な vSphere および TKG のバージョン

特に明記されていない限り、このドキュメントは vSphere IaaS control plane の vSphere 8 バージョンに適用され、スーパーバイザー 上の TKG v3.x をサポートするように更新されます。ドキュメントの更新は累積的です。以前のリリースのドキュメントのアーカイブは、VMware ドキュメント サイトの見出し [アーカイブ パッケージ] でダウンロードできます。

更新情報

1

このドキュメントは、必要に応じて新しい情報と修正を追加して定期的に更新されます。

このドキュメントの更新履歴については、次の表をご確認ください。

リビジョン	説明
2024 年 6 月 25 日	vSphere 8 Update 3 および TKG サービス 3.0 をサポートするようにドキュメントを更新しました。内容は以下のとおりです。 ■ 3 章 TKG サービスのインストールとアップグレード ■ 10 章 TKG サービス クラスタの自動スケーリング ■ 20 章 TKG サービス クラスタとワークロードのバックアップとリストア ■ TKG サービス クラスタに対する Antrea-NSX Adapter アダプタの有効化 ■ NSX 管理プロキシ サービスのインストール ■ 11 章 TKG サービス クラスタへの標準パッケージのインストール ■ v1beta1 クラスタの MachineHealthCheck の構成 ■ クラスタ v1beta1 API （更新） ■ controlPlaneCertificateRotation（更新済み） ■ podSecurityStandard (新規) ■ v1beta1 の例：カスタム ClusterClass に基づくクラスタ（vSphere 8 U2 以降のワークフロー）（更新） ■ TKG クラスタでの Velero Plugin for vSphere のインストールと構成（更新） ■ kubectl を使用したクラスタの手動スケーリング（更新）
2024 年 3 月 8 日	■ 最小バージョンが TKr v1.26 以降を使用する vSphere 8.0 Update 2 以降であることを示すように CSI スナップショットの要件を更新しました。15 章 TKG サービス クラスタでのスナップショットの作成を参照してください。 ■ CSI スナップショットで Velero を使用して TKGS クラスタ ワークロードをバッキングおよびリストアする手順を追加しました。Velero と CSI スナップショットを使用したバックアップとリストアを参照してください。 ■ FQDN を使用して v1beta1 API クラスタをプロビジョニングするための例と検証手順を更新しました。v1beta1 の例：FQDN を使用するクラスタを参照してください。 ■ vSphere 名前空間 の [表示可能] ロール権限の説明と Kubernetes の制限事項を更新しました。ロールの権限とバインドを参照してください。 ■ vCenter Server の信頼できるルート CA 証明書を Linux クライアント ホストにインストールする明示的な手順を追加しました。vCenter Server 用の信頼できるルート CA 証明書のダウンロードと Ubuntu クライアントへのインストールを参照してください。

リビジョン	説明
2024 年 2 月 29 日	■ vSphere 8 U2 P03 の GA リリースをサポートするようにドキュメントを更新しました。 ■ FQDN を使用したクラスタの作成に関するドキュメントを追加しました。クラスタ v1beta1 API と v1beta1 の例：FQDN を使用するクラスタを参照してください。 ■ FQDN が有効な場合に使用できることを示すようにスーパーバイザー ログオン コマンドを更新しました。 ■ 例を追加して PSA ドキュメントを更新しました。TKR 1.25 以降の PSA の構成を参照してください。 ■ TKGS ワークロード クラスタに NVIDIA Network Operator をインストールして vGPU ワークロードを実行する手順を追加しました。TKG サービス クラスタへの AI/ML ワークロードのデプロイに関するクラスタ オペレータのワークフローを参照してください。 ■ 仮想マシン クラス作成ウィザードの使用（具体的には、NVIDIA GRID vGPU 用のカスタム仮想マシン クラスの作成）に関するコンテンツを追加しました。13 章 TKG サービス クラスタへの AI/ML ワークロードのデプロイを参照してください。 ■ 「#unique_22」および「TKG クラスタでの Velero Plugin for vSphere のインストールと構成」を更新し、Velero のバージョン互換性マトリックスへのリンクを追加しました。
2024 年 2 月 2 日	■ vSphere 8 で v1alpha2 API が v1alpha3 API に自動変換されることを示すように TKG クラスタ プロビジョニングのドキュメントを更新しました。TKG クラスタのプロビジョニングについてを参照してください。
2024 年 1 月 31 日	■ その他のバグ修正および編集を行いました。
2024 年 1 月 19 日	■ TKGS クラスタとプライベート コンテナ レジストリの統合に関するトピックを更新しました。TKG サービス クラスタとプライベート コンテナ レジストリの統合を参照してください。
2024 年 1 月 18 日	■ 「クラスタ v1beta1 API 」を更新し、`defaultRegistrySecret` 変数が組み込みの Harbor レジストリで使用するために予約されていることを明示しました。
2024 年 1 月 16 日	■ TKG 標準パッケージ リポジトリ バージョン v2023.9.19 以降を使用する必要があることを示すように「15 章 TKG サービス クラスタでのスナップショットの作成」を更新しました。
2024 年 1 月 8 日	■ その他のマイナー更新を行いました。
2024 年 1 月 2 日	■ その他のマイナー更新を行いました。
2023 年 12 月 20 日	■ 「TKG サービス クラスタでの Kubernetes リリースの使用」を更新し、カスタム TKr イメージの構築に関するドキュメントへのリンクを追加しました。 ■ その他のマイナー更新を行いました。
2023 年 12 月 6 日	■ TKR リリース ノートを更新し、新しい Tanzu Kubernetes リリースを追加しました。 ■ システム名前空間の PSA を変更できないことを示すように「TKR 1.25 以降の PSA の構成」を更新しました。 ■ 「TKG サービス クラスタのローリング アップデート モデルについて」を更新し、複数のノード プールを使用する場合の考慮事項を追加しました。
2023 年 11 月 21 日	■ ルーティング可能なポッド ネットワークの最小サイズが /21 であることを示すように「v1alpha3 の例：ルーティング可能なポッド ネットワークを使用する TKC」および「v1beta1 の例：ルーティング可能なポッド ネットワークを使用するクラスタ」を更新しました。 ■ 「TKG サービス クラスタのローリング アップデート モデルについて」を更新し、vSphere 7 スーパーバイザー の更新のためにローリング アップデートがトリガされるタイミングを明示しました。 ■ TKr に PSP が必要なタイミングを指定するように「コンテナ デプロイ エラーのトラブルシューティング」を更新しました。
2023 年 11 月 13 日	■ 有効な構成キーを英数字、「-」、「_」、または「.」で構成する必要があることを示すようにドキュメントを更新しました（例：「key.name」、「KEY_NAME」、「key-name」）。

リビジョン	説明
2023 年 11 月 7 日	■ TKr バージョン文字列のアップグレードに関するトピックを更新しました。TKR バージョンの編集による TKG クラスタの更新を参照してください。
2023 年 10 月 16 日	■ リンクを修正しました。
2023 年 10 月 13 日	■ TKG クラスタのスケーリング手順を更新し、v1beta1 API の使用時の例を追加しました。kubectl を使用したクラスタの手動スケーリングを参照してください。
2023 年 10 月 11 日	■ Tanzu CLI を使用して TKG クラスタに OIDC ユーザーとして接続する手順を更新しました。Tanzu CLI を使用した OIDC ユーザーとしての TKG クラスタへの接続を参照してください。
2023 年 10 月 9 日	■ コンテンツ ライブラリを vSphere 名前空間 に関連付ける手順を更新しました。TKR コンテンツ ライブラリの TKG サービス への関連付けを参照してください。
2023 年 10 月 3 日	■ vSphere 8 U2 カスタム ClusterClass のドキュメントを更新し、管理対象外のカスタム ClusterClass を作成するための注釈を追加しました。v1beta1 の例 : カスタム ClusterClass に基づくクラスタ（vSphere 8 U2 以降のワークフロー）を参照してください。
2023 年 9 月 21 日	次に示す vSphere 8 U2 リリースの機能に関する内容を更新しました。 ■ TKG クラスタのボリューム スナップショットの作成。15 章 TKG サービス クラスタでのスナップショットの作成を参照してください。 ■ TKG クラスタへのパッケージのインストール。11 章 TKG サービス クラスタへの標準パッケージのインストールを参照してください。 ■ v1beta1 API を使用してカスタム ClusterClass クラスタをプロビジョニングする。クラスタ v1beta1 API の使用を参照してください。 ■ TKG クラスタのローリング アップデートに関する考慮事項。TKG サービス クラスタのローリング アップデート モデルについてを参照してください。 ■ メンテナンスに関するその他の内容の更新および編集。 **注：** vSphere IaaS control plane と Tanzu Kubernetes リリース については、「リリース ノート」を参照してください。
2023 年 8 月 16 日	■ ルーティング可能なポッドを使用する v1alpha3 TKC と、カスタム ClusterClass を使用する v1beta1 クラスタの TKG クラスタ プロビジョニングの例を更新しました。7 章 TKG サービス クラスタのプロビジョニングを参照してください。
2023 年 8 月 15 日	■ Tanzu パッケージのインストールのドキュメントを更新しました。11 章 TKG サービス クラスタへの標準パッケージのインストールを参照してください。 ■ 軽微な編集と修正。
2023 年 8 月 10 日	■ パブリック Harbor レジストリから スーパーバイザー サービスとして実行されているプライベート Harbor レジストリに Tanzu パッケージ イメージをプッシュするためのドキュメントを追加しました。14 章 TKG サービス クラスタでのプライベート レジストリの使用を参照してください。 ■ 軽微な編集と修正。
2023 年 8 月 7 日	■ サービス タイプ LoadBalancer with NSX 向けの ExternalTrafficPolicy および LoadBalancerSourceRanges のドキュメントを削除しました。これらの機能はサポートされていません。
2023 年 8 月 2 日	■ Tanzu CLI と kubectl を使用して TKG クラスタに接続するためのトピックを追加しました。#unique_36 を参照してください。 ■ 軽微な編集と修正。
2023 年 7 月 31 日	■ 軽微な編集と修正。
2023 年 7 月 27 日	■ TKG 2.2 のサポートに関する記述を追加してドキュメントを更新しました。

リビジョン	説明
2023 年 7 月 21 日	■ Tanzu Kubernetes リリース のコンテンツ ライブラリを管理するために必要な権限のリストを追加しました。
2023 年 7 月 12 日	■ 編集および表示される編集者ロールと閲覧者ロールの名前を更新しました。TKG サービス クラスタの ID およびアクセス権の管理についてを参照してください。
2023 年 7 月 10 日	■ TKG クラスタで使用できるコンテンツ ライブラリを スーパーバイザー で編集するための手順を追加しました。既存のコンテンツ ライブラリの編集を参照してください。 ■ スーパーバイザー 上の TKG クラスタのサポート バンドルの収集に関するトピックを更新しました。スーパーバイザー 上の TKG クラスタに対するトラブルシューティング実行時のログの取得を参照してください。
2023 年 7 月 7 日	■ スーパーバイザー 上の TKG 2.0 を使用してプロビジョニングできる 2 つの異なるクラスタ タイプについて説明するトピックを更新しました。TKG クラスタのプロビジョニングについてを参照してください。 ■ カスタム ノードのボリューム マウントを使用したクラスタのプロビジョニングに関する考慮事項を含むトピックを追加しました。ノード ボリューム マウントの使用に関する考慮事項を参照してください。
2023 年 7 月 3 日	■ vSphere IaaS control plane オペレータのグループとロールの作成に関するトピックを更新しました。プラットフォーム オペレータ専用のグループおよびロールの作成を参照してください。
2023 年 6 月 30 日	■ Cert Manager をインストールするためのスクリプトを更新しました。#unique_41 を参照してください。 ■ スーパーバイザー に Tanzu Mission Control Self-Managed のインスタンスを登録する手順を追加しました。Tanzu Mission Control Self-Managed の スーパーバイザー への登録を参照してください。 ■ Distributed Switch 環境の仮想マシン ジャンプ ホストに関する手順を更新しました。Linux ジャンプ ホスト仮想マシンの作成を参照してください。
2023 年 6 月 26 日	■ vSphere 8 スーパーバイザー でサポートされる項目に関する詳細を追加して TKG クラスタ API のリストを更新しました。TKG クラスタのプロビジョニングについてを参照してください。 ■ 専用の TKG クラスタ オペレータ ロールを作成するための権限リストを更新しました。プラットフォーム オペレータ専用のグループおよびロールの作成を参照してください。 ■ Tanzu CLI を使用して Contour をインストールするコマンドを更新しました。パッケージ リポジトリの作成を参照してください。
2023 年 6 月 13 日	■ 専用の TKG クラスタ オペレータ グループとロールの作成に関するトピックを追加しました。プラットフォーム オペレータ専用のグループおよびロールの作成を参照してください。 ■ ルーティング可能なポッド ネットワークを使用した v1beta1 クラスタの作成に関するトピックを更新しました。v1beta1 の例：ルーティング可能なポッド ネットワークを使用するクラスタを参照してください。 ■ 既知の問題により、ルーティング可能なポッド ネットワークを使用した v1alpha3 Tanzu Kubernetes クラスタの作成に関するトピックを削除しました。詳細については、「リリース ノート」を参照してください。
2023 年 6 月 9 日	■ スーパーバイザー からの vSphere 名前空間 の削除に関するトピックを追加しました（その vSphere 名前空間 でプロビジョニングされた TKG クラスタを先に削除するという前提条件も記載しました）。vSphere 名前空間 の削除を参照してください。
2023 年 6 月 6 日	■ ドキュメント セットを構成するさまざまなドキュメント間を移動するための vSphere IaaS control plane ドキュメント マップ グラフィックを追加しました。vSphere IaaS 制御プレーンでの TKG サービスの使用を参照してください。
2023 年 6 月 5 日	■ 2 章 TKG サービス クラスタの実行と 9 章 TKG サービス クラスタの更新の各セクションで編集上の変更を実施しました。 ■ 「vSphere with Tanzu Certificate Guide」への参照およびその他の変更を追加して「TKG サービス クラスタの TLS 証明書の管理」を更新しました。

リビジョン	説明
2023 年 6 月 1 日	■ クラスタのプロビジョニング例を更新しました。TanzuKubernetesCluster v1alpha3 API の使用とクラスタ v1beta1 API の使用を参照してください。 ■ クラスタのスケーリング例を更新しました。kubectl を使用したクラスタの手動スケーリングを参照してください。
2023 年 5 月 26 日	■ TKG クラスタのローリング アップデートに関するトピックを更新し、詳細を追加しました。TKG サービス クラスタのローリング アップデート モデルについてを参照してください。
2023 年 5 月 23 日	■ TKG 2 の代わりに TKG 2.0 を使用するようにドキュメントを編集しました。
2023 年 5 月 22 日	■ 名前空間付きポッドのセキュリティ ポリシーを作成するためのコマンドを更新しました。TKG サービス クラスタへのデフォルトのポッド セキュリティ ポリシーの適用を参照してください。 ■ アップグレード ワークフローのドキュメントを更新しました。#unique_51 を参照してください。
2023 年 5 月 12 日	■ 3 つの vSphere Zone にデプロイされた スーパーバイザー インスタンスにのみ新しい TKG 2 クラスタをプロビジョニングできるという注記を追加しました。
2023 年 5 月 11 日	■ パーシステント ボリュームの拡張に関するトピックを追加しました。TKG サービス クラスタのパーシステント ボリュームの拡張を参照してください。
2023 年 5 月 9 日	■ この例の複数のゾーンにまたがってポッドがデプロイされていることを示すために、ゾーン化されたトポロジのグラフィックを更新しました。TKG サービス クラスタのリファレンス アーキテクチャを参照してください。 ■ 「TKG サービス クラスタでの vSphere 名前空間 の使用」を更新し、ストレージの詳細を追加しました。 ■ 「8 章 TKG サービス クラスタの操作」を更新し、パーシステント ストレージの監視の詳細を追加しました。
2023 年 5 月 4 日	■ `--package` を使用するように「パッケージ リポジトリの作成」の内容を更新しました。 ■ 「TKR 1.24 以前の PSP の構成」を更新し、TKG 2 クラスタに PSP が引き続き必要であること、およびポッド セキュリティ アドミッション コントローラがサポートされていないことを明示しました。 ■ 「#unique_22」を更新しました。
2023 年 5 月 1 日	■ リンクを修正しました。
2023 年 4 月 4 日	■ vSphere 8 Update 1 リリースに一般的な更新と機能強化を適用。 ■ vSphere 8 U1 リリースの TKG サービス クラスタでの Kubernetes リリースの使用を更新しました。これには、スーパーバイザー および TKG 2 との TKR の互換性の確認、TKR NAME 文字列からの `-zshippable` サフィックスの削除、独自の TKR の構築が含まれます。 ■ クラスタ v1beta1 API の仕様に関するドキュメントを更新しました。 ■ TKG サービス クラスタとプライベート コンテナ レジストリの統合を更新して、例を追加しました。 ■ kubectl のテキスト エディタの構成を更新して、Windows の例を追加しました。 ■ バグを修正して スタンドアローンの Velero と Restic を使用した スーパーバイザー での TKG クラスタ ワークロードのバックアップとリストア を更新しました。

TKG サービス クラスタの実行

2

このセクションでは、コンポーネントについて説明し、TKG サービス クラスタを実行するための要件を示します。

次のトピックを参照してください。

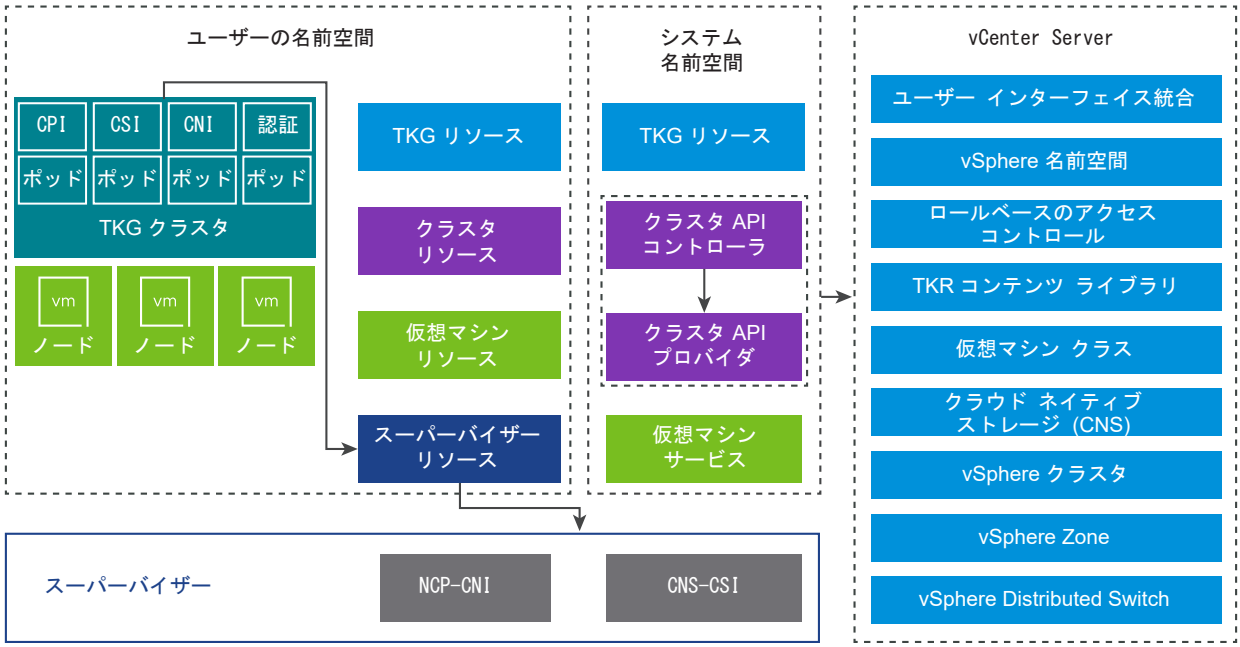
- TKG サービス コンポーネント
- TKG サービス クラスタのデプロイ
- TKG サービス クラスタのリファレンス アーキテクチャ

TKG サービス コンポーネント

TKG サービス は、Kubernetes ワークロード クラスタのセルフサービス ライフサイクル管理を提供します。スーパーバイザー を使用する TKG サービスは、vSphere 環境用に最適化され、vCenter Server、ESXi、仮想ネットワーク、クラウド ネイティブ ストレージなど、基盤となるインフラストラクチャと統合されます。TKG サービスを使用して、適合する Kubernetes クラスタをプロビジョニングし、アップストリーム同時実行を維持できます。

TKG サービスには、vSphere IaaS control plane と統合されているいくつかのコンポーネントが含まれています。

図 2-1. TKG サービス コンポーネント



ワークロード管理

ワークロード管理は、ネイティブ vSphere インフラストラクチャ上で Kubernetes 制御プレーンを提供する VMware ソリューションです。ワークロード管理を有効にすると、vSphere 環境に 1 つ以上の スーパーバイザーをデプロイできるようになります。ワークロード管理は vCenter Server にバンドルされていますが、個別にライセンス供与されます。

スーパーバイザー

スーパーバイザー は、ワークロード クラスタを管理するための制御プレーンとして機能する Kubernetes クラスタです。Tanzu Kubernetes Grid クラスタをプロビジョニングおよび運用する開発者をサポートするよう スーパーバイザー を構成します。

スーパーバイザー のデプロイ：vSphere クラスタ

スーパーバイザー のデプロイは、従来の方法では単一の vSphere クラスタに対して行います。vSphere クラスタは、vCenter Server によって管理される ESXi ホストの集合です。vSphere クラスタでは、次のような、スーパーバイザー をサポートするための特定の機能を構成する必要があります。

- vSphere Distributed Switch (VDS) によって接続された複数の ESXi ホスト
- vSAN、NFS などの共有ストレージが構成されている
- vSphere HA と完全自動化された DRS が有効になっている
- Lifecycle Manager が有効になっている
- ネットワークが構成されている（ロード バランサが組み込まれた NSX、または外部ロード バランサを使用する Distributed Switch）

設計上、スーパーバイザー上の TKG の本番環境では、vCenter Server が実行されている管理プレーン ホストまたはクラスタを、スーパーバイザーが有効になるコンピューティング プレーン クラスタから分離する必要があります。vSphere クラスタを使用して vCenter Server をホストしている場合、そのクラスタで DRS を有効にすることはできません。詳細については、vCenter Server のドキュメントを参照してください。

スーパーバイザー のデプロイ：vSphere ゾーン

vSphere 8 では、vSphere Zone が導入されています。vSphere クラスタを vSphere Zone に割り当てて、スーパーバイザー に対して高可用性とフォルト トレランスを実現できます。複数の vSphere Zone に スーパーバイザー をデプロイすると、特定の可用性ゾーンに TKG クラスタをプロビジョニングできます。

単一の vSphere クラスタに スーパーバイザー をデプロイする場合、スーパーバイザー と vSphere クラスタの間には 1 対 1 の関係があります。ゾーン化された スーパーバイザー デプロイでは、スーパーバイザー が 3 つの vSphere クラスタに拡張され、スーパーバイザー 上の TKG クラスタに高可用性と障害ドメインが提供されます。

vSphere 管理者は 3 つの vSphere Zone を作成し、vSphere クラスタに関連付けます。3 つの vSphere Zone に スーパーバイザー をデプロイするには、vSphere クラスタの要件に加えて、次のような特別な要件があります。

- vSphere Distributed Switch (VDS) によって接続された 3 つの vSphere Zone がある
- 各 vSphere Zone に単一の vSphere クラスタが含まれている
- 3 つの vSphere Zone に スーパーバイザー をデプロイする必要がある
- vSphere ストレージ ポリシーを vSphere Zone で使用できる

vSphere 名前空間

vSphere 名前空間 は、1 つ以上の Tanzu Kubernetes Grid クラスタがプロビジョニングされている スーパーバイザー 上の名前空間です。それぞれの vSphere 名前空間 について、ロールベースのアクセス制御、パーシステント ストレージ、リソース制限、イメージ ライブラリ、および仮想マシン クラスを構成します。

TKG サービス

TKG サービス は、Kubernetes クラスタのライフサイクルを管理するための一連のカスタム リソースとコントローラを定義するオープン ソース クラスタ API プロジェクトの実装です。Tanzu Kubernetes Grid は、スーパーバイザー のコンポーネントです。

TKG サービス には、TKG クラスタのライフサイクルを管理するためのコントローラのレイヤーが 3 つあります。これには 仮想マシン サービス、クラスタ API、クラウド プロバイダ プラグインが含まれます。

仮想マシン オペレータ

仮想マシン サービス コントローラは、仮想マシンとそれに関連する vSphere リソースを管理するための、宣言型の Kubernetes 形式 API を提供します。仮想マシン サービス により、抽象的な再利用可能ハードウェア構成を表す仮想マシン クラスの概念が導入されます。TKG サービス は仮想マシン サービスを使用して、ワークロード クラスタをホストする制御プレーンおよびワーカー ノード仮想マシンのライフサイクルを管理します。

クラスタ API

クラスタ API コントローラは、クラスタを作成、構成、および管理するための、宣言型の Kubernetes 形式 API を提供します。クラスタ API への入力には、クラスタを記述するリソース、クラスタを構成する仮想マシンを記述するリソースのセット、クラスタのアドオンを記述するリソースのセットなどがあります。

クラウド プロバイダ プラグイン

TKG サービス は、基盤となる vSphere 名前空間 リソースと統合するために必要なコンポーネントを含むワークロード クラスタをプロビジョニングします。これらのコンポーネントには、スーパーバイザー と連携するクラウド プロバイダ プラグインが含まれています。TKG はクラウド プロバイダ プラグインを使用して、VMware クラウド ネイティブ ストレージ (CNS) と統合されている スーパーバイザー にパーシステント ボリュームの要求を渡します。

Tanzu Kubernetes リリース

Tanzu Kubernetes リリース には、Tanzu Kubernetes Grid クラスタで使用するために VMware によって署名され、サポートされている Kubernetes ソフトウェア ディストリビューションとアドオンが含まれます。

各 Tanzu Kubernetes リリース は、仮想マシン テンプレート（OVA ファイル）として配布されます。Tanzu Kubernetes Grid は OVA 形式を使用して、TKG クラスタのための仮想マシン ノードを構築します。Tanzu Kubernetes リリースは、Kubernetes のバージョン管理に従ってバージョン管理され、vSphere インフラストラクチャの OS のカスタマイズと最適化が含まれます。

Tanzu Kubernetes リリースのリストおよび スーパーバイザー との互換性については、Tanzu Kubernetes リリースのリリース ノートを参照してください。vSphere IaaS control plane サポート ポリシーも参照してください。

コンテンツ ライブラリ

Tanzu Kubernetes リリース は、vCenter Server コンテンツ ライブラリを使用すると、TKG クラスタで利用できるようになります。サブスクライブ済みコンテンツ ライブラリを作成して、VMware によって利用可能にされた TKR を自動的に受信することができます。また、ローカル コンテンツ ライブラリを使用して、TKR を手動でアップロードすることもできます。

TKG サービス クラスタ コンポーネント

TKG サービス クラスタで実行されるコンポーネントは、製品の 4 つの領域（認証、ストレージ、ネットワーク、ロード バランシング）の機能を提供します。

認証 Webhook

認証 Webhook はクラスタ内のポッドとして動作し、ユーザー認証トークンを検証します。

TKG クラスタは、vCenter Single Sign-On の使用、および Open ID Connect (OIDC) プロトコルをサポートする外部 ID プロバイダの使用という 2 つの認証方法をサポートしています。

TKG は、スーパーバイザー 上および TKG クラスタ ノード上で Pinniped OIDC クライアントを実行します。スーパーバイザー の外部 OIDC プロバイダを構成すると、Pinniped コンポーネントが自動的に構成されます。

TKG は、kubectl 向けの vSphere プラグイン や Tanzu CLI など、スーパーバイザー を使用したさまざまな認証用クライアントをサポートします。

コンテナ ストレージ インターフェイス (CSI)

準仮想化 CSI プラグインは、TKG クラスタ内で実行され、スーパーバイザー を介して VMware 上のクラウド ネイティブ ストレージ (CNS) と統合される Kubernetes ポッドです。TKG クラスタ内で実行される Kubernetes ポッドは、短期、パーシステント ボリューム、コンテナ イメージという 3 種類の仮想ディスクをマウントできます。

[一時的なストレージ]

ポッドでは、ログ、ボリューム、構成マップのような Kubernetes オブジェクトなどの短期データを保存するために、一時的なストレージが必要です。一時的なストレージは、ポッドが存在する間は保持されます。短期データはコンテナの再起動後も維持されますが、ポッドが削除されると、短期データを保存していた仮想ディスクは破棄されます。

[パーシステント ストレージ]

TKG は、vSphere ストレージ ポリシー フレームワークを利用して、ストレージ クラスを定義し、パーシステント ボリュームを予約します。TKG クラスタは、VMware クラウド ネイティブ ストレージ (CNS) と統合されている スーパーバイザー にパーシステント ボリュームの要求を渡します。パーシステント ボリュームは、ストレージ クラスを使用して動的に、または手動で、クラスタ ノードにプロビジョニングされます。

[コンテナ イメージ ストレージ]

Kubernetes ポッド内のコンテナでは、実行するソフトウェアを含むイメージを使用します。ポッドは、コンテナで使用されるイメージをイメージ仮想ディスクとしてマウントします。ポッドのライフサイクルが完了すると、イメージ仮想ディスクはポッドから接続を解除されます。Kubelet は、イメージ レジストリからコンテナ イメージをプルし、そのイメージをポッド内で実行される仮想ディスクに変換します。

コンテナ ネットワーク インターフェイス (CNI)

コンテナ ネットワーク インターフェイス プラグインは、ポッド ネットワークを提供する CNI プラグインです。

TKG クラスタでは、Antrea（デフォルト）と Calico という コンテナ ネットワーク インターフェイス (CNI) オプションがサポートされています。さらに、TKG は ルーティング可能なポッド ネットワークを実装するための Antrea NSX Routed CNI を提供します。

次の表に、TKG クラスタのネットワーク機能とその実装の概要を示します。

表 2-1. TKG サービス クラスタ ネットワーク

エンドポイント	プロバイダ	説明
ポッドの接続	Antrea または Calico	ポッドのコンテナ ネットワーク インターフェイス。Antrea は Open vSwitch を使用します。Calico は BGP を利用する Linux ブリッジを使用します。
サービス タイプ：ClusterIP	Antrea または Calico	クラスタ内からのみアクセス可能なデフォルトの Kubernetes サービス タイプ。

表 2-1. TKG サービス クラスタ ネットワーク （続き）

エンドポイント	プロバイダ	説明
サービス タイプ：NodePort	Antrea または Calico	Kubernetes ネットワーク プロキシによって各ワーカー ノードで開かれているポートを介して外部からアクセスできるようにします。
ネットワーク ポリシー	Antrea または Calico	選択したポッドとネットワーク エンドポイントの間で送受信されるトラフィックを制御します。Antrea は Open vSwitch を使用します。Calico は Linux IP テーブルを使用します。

クラウド プロバイダの実装

クラウド プロバイダの実装を使用すると、Kubernetes ロード バランサ サービスと Ingress サービスを作成できます。

表 2-2. TKG ロード バランシング

エンドポイント	プロバイダ	説明
サービス タイプ：LoadBalancer	NSX 組み込みロード バランサ（NSX ネットワーク スタックの一部） NSX Advanced Load Balancer（VDS ネットワークで使用するための個別のインストール） HAProxy（VDS ネットワークで使用するための個別のインストール）	NSX 組み込みロード バランサの場合、サービス タイプの定義ごとに 1 台の仮想サーバ。 NSX Advanced Load Balancer については、このドキュメントの該当するセクションを参照してください。 HAProxy については、このドキュメントの該当するセクションを参照してください。 **注：** サポートされている各ロード バランサ タイプで、固定 IP アドレスなど、一部のロード バランシング機能を使用できない場合があります。
クラスタの Ingress	サードパーティ製の Ingress コントローラ	受信ポッド トラフィックのルーティングを提供します。Contour など、サードパーティ製の Ingress コントローラを使用できます。

TKG サービス クラスタ API

TKG サービス では、TKG クラスタのライフサイクルをプロビジョニングおよび管理するための API が 2 つ提供されています。

- Tanzu Kubernetes クラスタ用の API バージョン `v1alpha3`
- ClusterClass に基づくクラスタ用の API バージョン `v1beta1`

v1alpha3 API を使用すると、`TanzuKubernetesCluster` タイプの適合する Kubernetes クラスタを作成できます。このタイプのクラスタは、迅速にプロビジョニングできるように一般的なデフォルト設定で事前構成されており、カスタマイズできます。v1beta1 API を使用すると、VMware によって提供されるデフォルトの `ClusterClass` に基づく、`Cluster` タイプの適合する Kubernetes クラスタを作成できます。

注： vSphere IaaS control plane を vSphere 7 から vSphere 8 にアップグレードするには、TKG クラスタで v1alpha2 API が実行されている必要があります。v1alpha2 API は v7.0 Update 3 で導入されました。v1alpha1 API は廃止されました。詳細については、「#unique_51」を参照してください。

TKG サービス クラスタのクライアント

vSphere 8 スーパーバイザー 上の TKG は、TKG クラスタをプロビジョニング、監視、および管理するためのさまざまなクライアント インターフェイスをサポートしています。

- スーパーバイザー の構成とデプロイされた TKG クラスタの監視のための vSphere Client。
- vCenter Single Sign-On を使用する スーパーバイザー および TKG クラスタでの認証のための kubectl 向けの vSphere プラグイン。
- TKG クラスタのライフサイクルを宣言によってプロビジョニングおよび管理し、スーパーバイザー と連携する kubectl。
- コンテナ レジストリとの間でイメージをプッシュおよびプルする vSphere Docker 認証情報ヘルパー。
- コマンドを使用してクラスタをプロビジョニングしたり、Tanzu パッケージをインストールしたりするための Tanzu CLI。
- TKG クラスタを管理するための Tanzu Mission Control Web インターフェイス。

TKG サービス クラスタのデプロイ

TKG サービス クラスタをデプロイするには、TKG サービス をインストールまたはアップグレードし、コンテンツ ライブラリを使用して Tanzu Kubernetes リリースを保存し、TKG クラスタをホストするための vSphere 名前空間を 1 つ以上構成します。

TKG サービス クラスタのデプロイ

TKG サービス クラスタをデプロイするには、[ワークロード管理] 有効化プロセスを完了し、スーパーバイザー インスタンスを構成します。

注： 手順については、『vSphere IaaS 制御プレーンのインストールと構成』を参照してください。

[ワークロード管理] を有効にして スーパーバイザー をデプロイしたら、TKG サービス クラスタのプロビジョニングの準備として次のタスクを実行します。

タスク	説明
TKG サービス のインストールまたはアップグレード	3 章 TKG サービスのインストールとアップグレードを参照してください。
CLI のインストールと スーパーバイザー への接続	4 章 TKG サービス クラスタの ID とアクセスの構成を参照してください。
Tanzu Kubernetes リリース (TKr) 用のコンテンツ ライブラリの作成と同期	コンテンツ ライブラリでは、TKG 環境と、TKr を配信する VMware コンテンツ配信ネットワークとが統合されます。要件に応じて、サブスクライブ済みまたはローカルのコンテンツ ライブラリを使用できます。 新しい TKG 機能を利用するには、既存の TKr をアップグレードする必要があります。5 章 TKG サービス クラスタ用 Kubernetes リリースの管理を参照してください。
TKG サービス クラスタをホストするための vSphere 名前空間 の作成と構成	vSphere 名前空間 を定義すると、TKG サービス クラスタをホストするためのネットワーク セグメントが作成されます。6 章 TKG サービス クラスタをホストするための vSphere 名前空間 の構成を参照してください。 各 vSphere 名前空間 は、以下を使用して構成する必要があります。 ■ クラスタのユーザーとロール ■ 同期された TKr コンテンツ ライブラリ ■ 関連付けられている仮想マシン クラス ■ TKG サービス クラスタ ノードとパーシステント ボリュームの vSphere ストレージ ポリシー 3 つの vSphere Zones にデプロイされたスーパーバイザーを使用している場合は、vSphere ストレージ ポリシーで `Zonal` トポロジを指定する必要があります。TKG サービス クラスタの vSphere ストレージ ポリシーの作成を参照してください。
TKG サービス クラスタのプロビジョニング	7 章 TKG サービス クラスタのプロビジョニングを参照してください。
TKG サービス クラスタの運用	8 章 TKG サービス クラスタの操作を参照してください。
クラスタ ワークロードのデプロイ	12 章 TKG サービス クラスタへのワークロードのデプロイを参照してください。
TKG サービス クラスタの保守	9 章 TKG サービス クラスタの更新を参照してください。
TKG サービス クラスタのトラブルシューティング	21 章 TKG サービス クラスタのトラブルシューティングを参照してください。

TKG サービス クラスタのリファレンス アーキテクチャ

このトピックでは、さまざまなデプロイ トポロジを使用した TKG サービス クラスタに対する一連のリファレンス アーキテクチャについて説明します。

NSX トポロジを使用した TKG サービス クラスタ

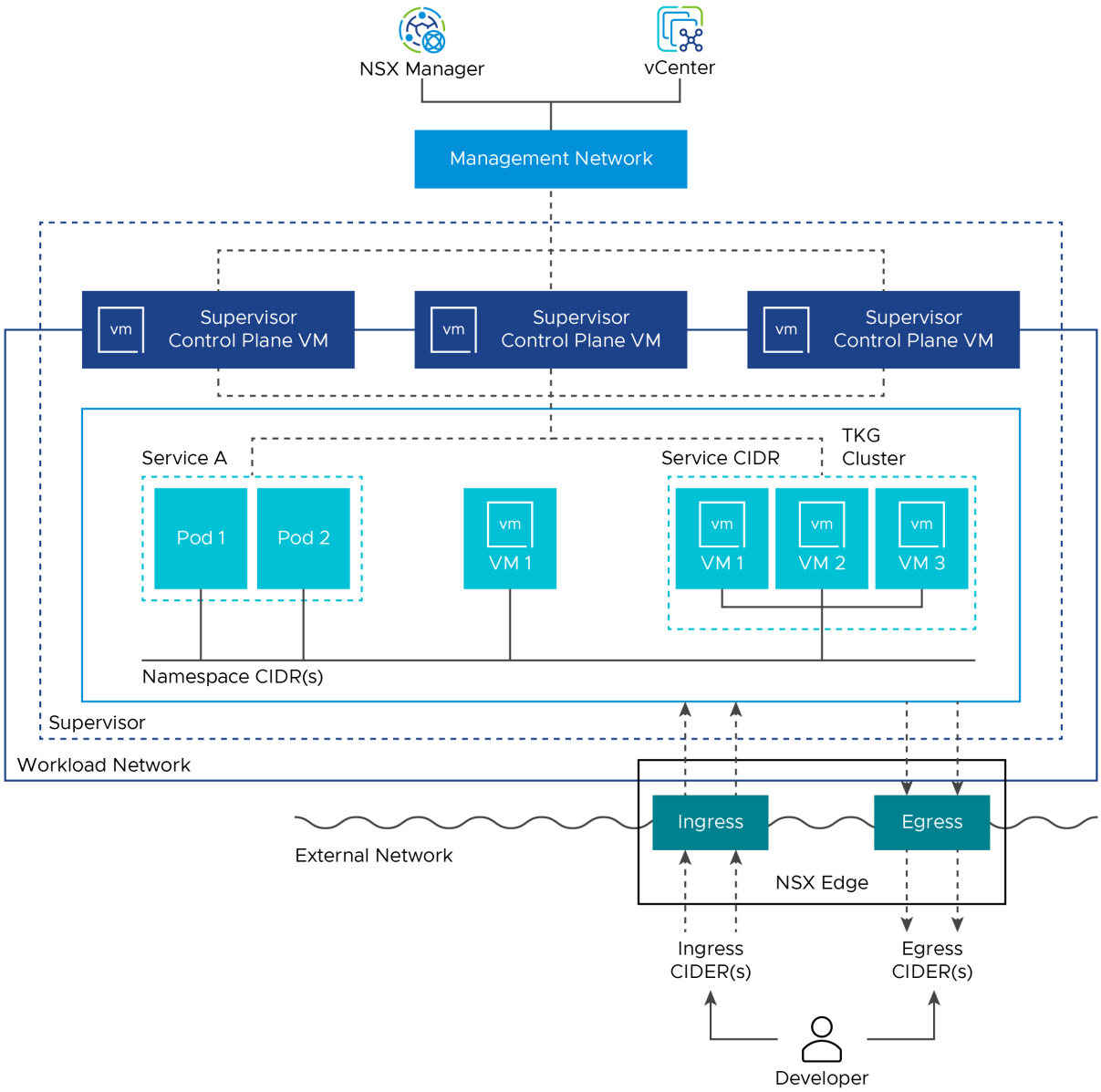
リファレンス アーキテクチャは、NSX ネットワークを使用する スーパーバイザー 上の TKG クラスタを示しています。このような環境では、管理プレーンは vCenter Server と NSX Manager をホストします。コンピューティング プレーンは NSX Edge ノードと スーパーバイザー ノードをホストします。

次の NSX ネットワーク オブジェクトがあります。

- Tier-1 ゲートウェイ（ルーター）

- ゲートウェイにリンクされたセグメント（スイッチ）
- ロード バランサ サーバ
- 各 TKG クラスタ制御プレーン仮想サーバのためのサーバ プール
- 各 Kubernetes サービス ロード バランサ インスタンスのための仮想サーバ

図 2-2. NSX トポロジを使用した TKG サービス クラスタ



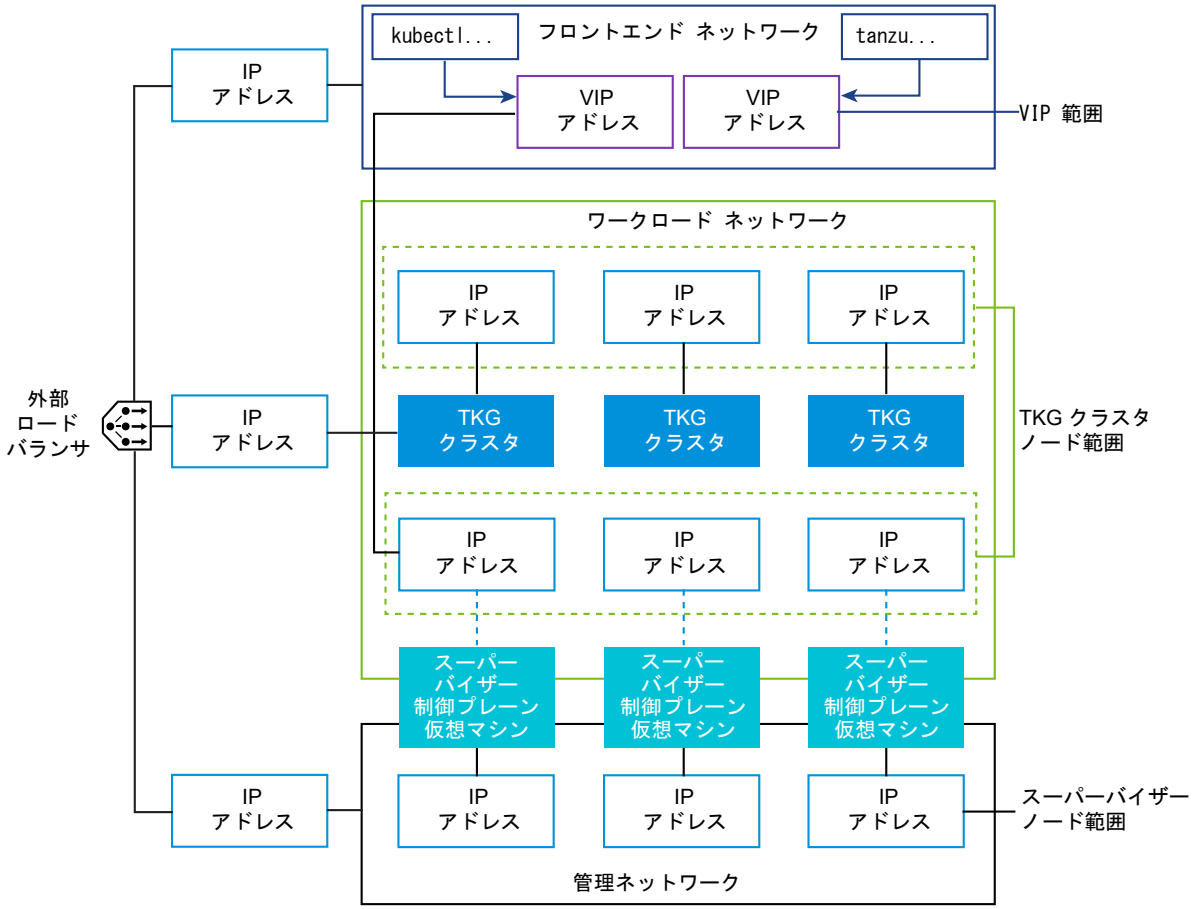
VDS トポロジを使用した TKG サービス クラスタ

リファレンス アーキテクチャは、VDS ネットワークと外部ロード バランサを使用する TKG サービス クラスタを示しています。このような環境には、次のネットワークがあります。

- スーパーバイザー 制御プレーン仮想マシンのための管理ネットワーク

- TKG クラスタのためのワークロード ネットワーク
- 開発者が TKG サービス クラスタに接続するためのフロントエンド ネットワーク

図 2-3. VDS トポロジを使用した TKG サービス クラスタ



vSphere ゾーン トポロジを使用した TKG サービス クラスタ

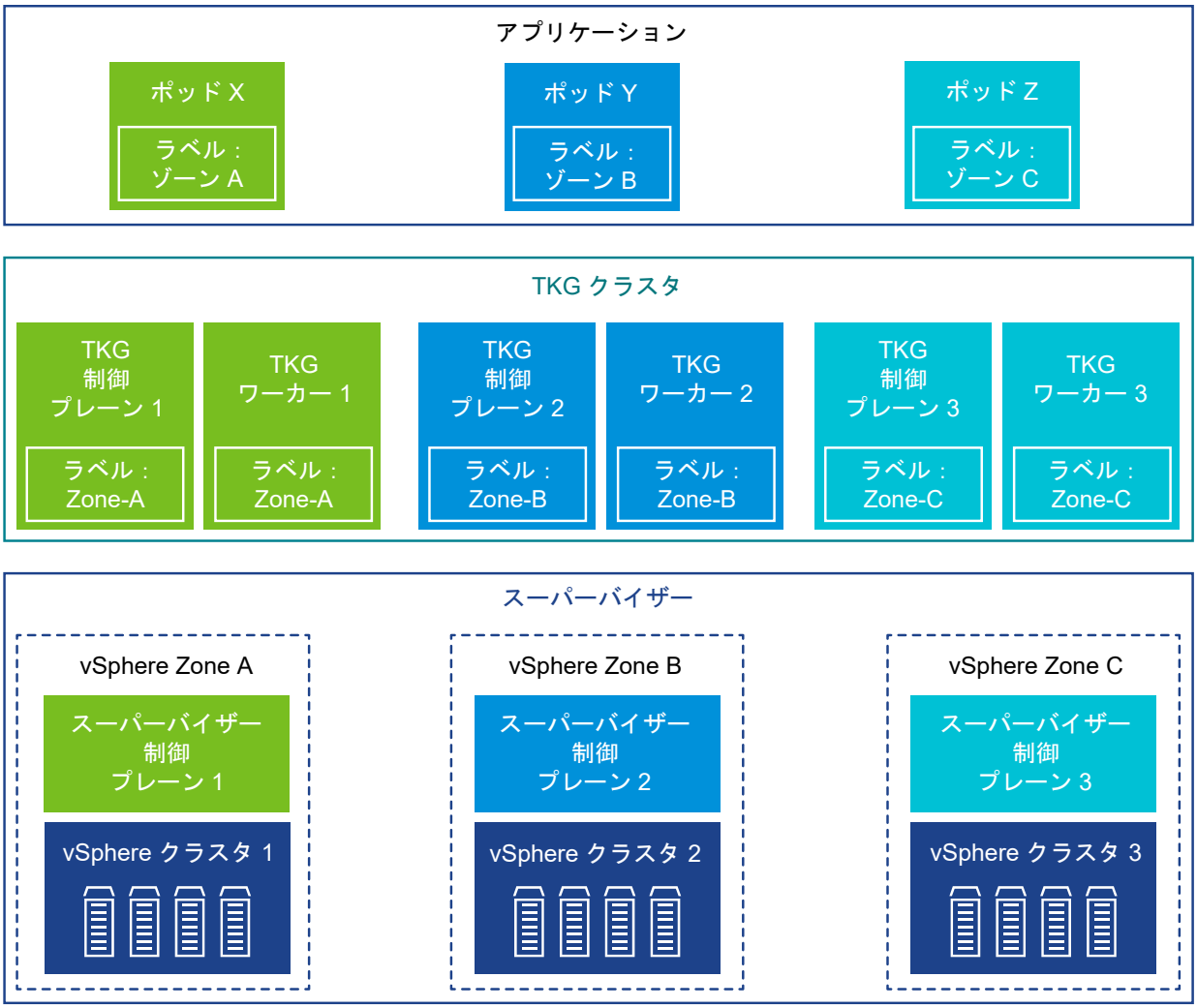
リファレンス アーキテクチャは、vSphere Zones 全体にデプロイされた TKG クラスタを示しています。各 vSphere Zone は、vSphere クラスタにマッピングされています。このクラスタは、vCenter Server によって管理され、vSphere Distributed Switch によって接続されていて、共有ストレージとサービスの品質が有効になっている ESXi ホストの集合です。

ゾーン トポロジでは、3 つの vSphere Zones にわたって スーパーバイザー をデプロイします。各 vSphere ゾーンに スーパーバイザー 制御プレーンが配置され、これにより障害発生時の高可用性が提供されます。

TKG クラスタを スーパーバイザー にプロビジョニングすると、クラスタでは vSphere Zones が認識されます。ゾーン トポロジでは、高可用性ワークロードのために、障害ドメインがサポートされます。必要に応じて、注釈を使用して特定のゾーンでワークロードを実行できます。

注： vSphere 8 では、vSphere ゾーンが新しくなっています。そのため、vSphere ゾーンでは、スーパーバイザー と TKG クラスタの新しいデプロイのみがサポートされています。

図 2-4. vSphere ゾーン トポロジを使用した TKG サービス クラスタ



TKG サービスのインストールとアップグレード

3

このセクションでは、TKG サービスのインストールとアップグレードについて説明します。

次のトピックを参照してください。

- TKG サービス の使用
- TKG サービス のステータスの確認
- 新しいバージョンの TKG サービス の登録
- TKG サービス バージョンのアップグレード
- TKG サービスのトラブルシューティング

TKG サービス の使用

VMware Tanzu Kubernetes Grid サービス (TKG サービス) を使用すると、vSphere IaaS control plane に Kubernetes ワークロード クラスタをデプロイできます。TKG サービス が提供するリリースは独立しており、ワークロードを中断することなく非同期アップグレードを行うことができます。

TKG サービス の概要

vSphere 8.0 Update 3 以降では、Tanzu Kubernetes Grid は スーパーバイザー サービス としてインストールされます。このアーキテクチャの変更により、TKG は vSphere IaaS control plane リリースから分離され、vCenter Server や スーパーバイザー に関係なく TKG サービス をアップグレードできます。

TKG サービス 3.0 がインストールされ、スーパーバイザー 制御プレーン ノードで実行されます。TKG サービス は、Carvel パッケージをネストしたコレクションとして提供されます。TKG サービス は、インターネットが制限されている環境でもコア スーパーバイザー サービス としてアップグレードできますが、アンインストールやダウングレードを実行することはできません。[ワークロード管理] - [サービス] タブで TKG サービス を監視および管理できます。TKG サービス バージョンのアップグレードを参照してください。

TKG サービス バージョン 3.1 は、ユーザーのアップグレード先となる最初の独立したリリースです。TKG サービス vCenter Server への新しい TKG サービス バージョンの登録と TKG サービス バージョンのアップグレードは個別のプロセスです。

TKG サービス 3.0 のインストール

vSphere IaaS control plane コンポーネントを必要なバージョンにアップグレードすると、TKG サービス のインストールが自動的に実行されます。詳細については、TKG サービス のリリース ノートを参照してください。

vCenter Server への新しい TKG サービス バージョンの登録

TKG サービス パッケージは vCenter Server によって公開され、VMware パブリック レジストリにプッシュされます。TKG サービス の登録は、vCenter Server レベルで実行されます。TKG サービス の新しいバージョンを登録するには、同期と非同期の 2 つのオプションがあります。

表 3-1. TKG サービス バージョンの登録オプション

登録方法	説明
同期	最新の vCenter Server リリースへの更新により、TKG サービス の新しいバージョンが自動登録されるまで待機してから、スーパーバイザー を更新し、組み込みレジストリに新しいバージョンをポピュレートします。
非同期	パブリック レジストリから新しい TKG サービス バージョンの定義をダウンロードし、vCenter Server に手動で登録します。

同期登録を行うは、システム更新が必要です。vCenter Server を更新すると、新しい TKG サービス バージョンが スーパーバイザー に自動的に登録されます。ただし、自動登録された（新しい）バージョンを使用するには、スーパーバイザー を、この vCenter Server によって提供される vSphere 名前空間 リリースに付属するバージョンに更新する必要があります。スーパーバイザー を更新すると、スーパーバイザー 組み込みレジストリで TKG サービス の Carvel パッケージ バンドルが使用可能になり、展開する準備が整います。スーパーバイザー をアップグレードしても、TKG サービス は自動的にアップグレードされません。必要なバージョンを選択してデプロイする必要があります。

非同期登録では、現在の スーパーバイザー バージョンがサポート ウィンドウ内にある場合、vCenter Server および スーパーバイザー の更新は必要ありません。非同期登録のワークフローは、次のとおりです。

1 スーパーバイザー サービス のパブリック レジストリ サイトからサービス定義 YAML ファイルをダウンロードします。

2 サービス定義を vCenter Server にアップロードして、TKG サービス の新しいバージョンを登録します。

次の表に、TKG サービス の登録の詳細を示します。

表 3-2. TKG サービス バージョンの登録

TKG サービスのプロパティ	バンドルされている vCenter Server	パブリック レジストリ
新しいバージョンの登録	自動登録	手動登録
新しく登録されたバージョンの削除	許可されていません	許可
イメージの場所	スーパーバイザー制御プレーンの組み込みレジストリ	パブリック レジストリ

TKG サービス バージョンのアップグレード

TKG サービス バージョンのアップグレードは、スーパーバイザー レベルで実行されます。TKG サービス が登録されたら、TKG サービス をターゲット スーパーバイザー に スーパーバイザー サービス としてデプロイして、アップグレードします。TKG サービス バージョンのアップグレードを参照してください。

インターネットが制限されている（エアギャップ）環境で TKG サービス をアップグレードするには、vCenter Server をアップデートして新しい TKG サービス バージョンを同期的に登録します。インストールするバージョンを選択すると、ローカル レジストリを使用して新しい TKG サービス バージョンがインストールされます。vCenter Server への新しい TKG サービス バージョンの登録を参照してください。

TKG サービス バージョンをアップグレードすると、システムは事前チェックを実行し、次の 2 段階の重要度を報告します。

- 警告（非ブロック）
- エラー（ブロック）

Kubernetes バージョン チェックは、ブロックを伴わない警告チェックの一例です。スーパーバイザー バージョン チェックは、ブロックを伴うエラーの一例です。詳細については、スーパーバイザー サービスのドキュメントを参照してください。

TKG サービス のステータスの確認

TKG サービス のステータスを確認するには、このトピックを参照してください。

このタスクを完了して、TKG サービス がコア スーパーバイザー サービス としてインストールされていることを確認し、そのステータスを確認します。

ステータスを確認する手段として、vSphere Client を使用する方法と kubectl を使用する方法の 2 つがあります。vSphere Client を使用してステータスを確認するには、vCenter Server にログインし、[ワークロード管理] - [サービス] の順に選択します。

kubectl を使用してステータスを確認するには、次の手順を実行します。

前提条件

このタスクでは、すべてのシステム コンポーネントをアップグレード済みであること、および TKG サービス 3.0 をインストール済みであることを前提にしています。「#unique_67/unique_67_Connect_42_TABLE_CED10728-0714-4D00-BF58-D4BBAA2A4D8E」を参照してください。

手順

1 kubectl を使用して スーパーバイザー にログインします。

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

2 次のコマンドを実行します。

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

TKG サービス がインストールされていることが表示されます。

```
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
```

新しいバージョンの TKG サービス の登録

TKG サービス を非同期にアップグレードする目的で、新しいバージョンの TKG サービス を vCenter Server に手動で登録するには、このトピックを参照してください。

このタスクは、TKG サービス バージョンを非同期にアップグレードする場合にのみ必要です。vCenter Server への新しい TKG サービス バージョンの登録を参照してください。

前提条件

このタスクでは、すべてのシステム コンポーネントをアップグレード済みであること、および TKG サービス 3.0 をインストール済みであることを前提にしています。「TKG サービス 3.0 のインストール」を参照してください。

手順

1. ブラウザを使用して、スーパーバイザー サービス の配布サイト (https://www.vmware.com/go/supervisor-service) に移動します。
2. サイトから TKG サービス の `package.yaml` ファイルをダウンロードします。
3. vSphere Client を使用して、vCenter Server にログインします。
4. [ワークロード管理] - [サービス] の順に移動します。
5. [Tanzu Kubernetes Grid Service] という名前のサービス タイルを見つけます。
6. [アクション] - [新規バージョンの追加] の順に選択します。
7. [アップロード] をクリックします。
8. ダウンロードした `package.yaml` ファイルを選択します。
9. [終了] をクリックします。

結果

新しいサービス定義を登録すると、TKG サービス タイルで使用可能な TKG サービス のバージョンが複数表示されることがあります。TKG サービス をアップグレードするときに、ターゲット バージョンを選択します。

次のステップ

TKG サービス バージョンのアップグレード。

TKG サービス バージョンのアップグレード

TKG サービス のバージョンをアップグレードするには、このトピックを参照してください。

次の手順を完了して、TKG サービス バージョンをアップグレードします。

TKG サービス バージョンのアップグレードは、スーパーバイザー レベルで実行されます。vCenter Server に複数の スーパーバイザー がホストされている場合は、ターゲット スーパーバイザー を選択する必要があります。

前提条件

このタスクでは、すべてのシステム コンポーネントをアップグレード済みであること、および TKG サービス 3.0 をインストール済みであることを前提にしています。「TKG サービス 3.0 のインストール」を参照してください。

このタスクでは、TKG サービス の新しいバージョンを新しいバージョンの TKG サービス の登録で、または vCenter Server への新しい TKG サービス バージョンの登録登録していることが前提となります。

手順

1 vSphere Client を使用して、vCenter Server にログインします。

2 [ワークロード管理] - [サービス] の順に移動します。

3 [Tanzu Kubernetes Grid Service] タイルを特定します。

4 [アクション] - [スーパーバイザーへのインストール] の順に選択します。

5 アップグレード先となる TKG サービス のターゲット バージョンを選択します。

6 アップグレードする TKG サービス をホストしているターゲット スーパーバイザー を選択します。

7 サービスの互換性を選択し、[OK] をクリックします。

8 TKG サービス がアップグレードされていることを確認します。

[ワークロード管理] - [サービス] 画面の [Tanzu Kubernetes Grid Service] タイルには、バージョンとステータスが反映されます。`kubectl get tkr` を使用してステータスを確認することもできます。

TKG サービスのトラブルシューティング

TKG サービスをトラブルシューティングするには、このトピックを参照してください。

TKG サービス サポート バンドル

TKG サービス サポート バンドルは、スーパーバイザー サポート バンドルに含まれています。ガイダンスについては、「スーパーバイザーのサポート バンドルの収集」を参照してください。

スーパーバイザー サポート バンドル内では、TKG サービス ログは `var/logs/tkg-svs` フォルダにあります。

スーパーバイザー サービスはコア サービス コントローラによって管理されます。コア サービス コントローラのログは `/var/log/vmware/wcp/` にあります。

アプリケーション プラットフォームのログは、次のコマンドを使用して取得できます。

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

TKG サービス クラスタの ID とアクセスの構成

4

このセクションでは、スーパーバイザー 上の TKG サービス クラスタのユーザー認証と認可の構成について説明します。

次のトピックを参照してください。

- TKG サービス クラスタの ID およびアクセス権の管理について
- TKG サービス クラスタ用 CLI ツールのインストール
- vCenter SSO 認証を使用した TKG サービス クラスタへの接続
- 外部 ID プロバイダを使用した スーパーバイザー 上の TKG クラスタへの接続
- Kubernetes 管理者およびシステム ユーザーとしての TKG サービス クラスタへの接続
- プラットフォーム オペレータ専用のグループおよびロールの作成

TKG サービス クラスタの ID およびアクセス権の管理について

DevOps エンジニアは、vSphere IaaS control plane に接続して TKG サービス クラスタのライフサイクルをプロビジョニングし、管理します。開発者は TKG サービス クラスタに接続して、パッケージ、ワークロード、およびサービスをデプロイします。管理者がトラブルシューティングを行うには、TKG サービス クラスタ ノードへの直接アクセスが必要になる場合があります。このプラットフォームには、ID とアクセス管理について、それぞれのユースケースとロールをサポートするツールと方法が用意されています。

TKG サービス クラスタ アクセスの対象範囲は vSphere 名前空間

vSphere 名前空間 で TKG サービス クラスタをプロビジョニングします。vSphere 名前空間 を構成する場合は、ID ソース、ユーザーとグループ、ロールも含めて、その名前空間の DevOps 権限を設定します。このプラットフォームでは、これらの権限を該当する vSphere 名前空間 でプロビジョニングされた各 TKG サービス クラスタに伝達します。このプラットフォームでサポートされている 2 つの認証方法は、vCenter Single Sign-On と OIDC 準拠の外部 ID プロバイダです。

vCenter Single Sign-On と Kubectl を使用した認証

デフォルトでは、vCenter Single Sign-On は、スーパーバイザー と TKG サービス クラスタを含む環境で認証を行うために使用されます。vCenter Single Sign-On は、vSphere インフラストラクチャに認証を提供し、AD/LDAP システムと統合できます。詳細については、「vCenter Single Sign-On による vSphere 認証」を参照してください。

vCenter Single Sign-On を使用して認証するには、kubectl 向けの vSphere プラグイン を使用します。認証されたら、kubectl を使用して、TKG サービス クラスタの宣言によるプロビジョニングと管理、および スーパーバイザー の操作を実行します。

kubectl 向けの vSphere プラグイン は kubectl に依存しています。`kubectl vsphere login` コマンドで認証すると、プラグインは スーパーバイザー の /wcp/login エンドポイントに対する基本認証を使用して、POST を実行します。vCenter Server は、スーパーバイザー が信頼する JSON Web トークン (JWT) を発行します。

vCenter Single Sign-On を使用して接続するには、vCenter SSO 認証を使用した TKG サービス クラスタへの接続を参照してください。

外部 ID プロバイダと Tanzu CLI を使用した認証

OpenID Connect プロトコルをサポートする外部 ID プロバイダを使用して スーパーバイザー を構成できます。構成された スーパーバイザー は OAuth 2.0 クライアントとして機能し、Pinniped 認証サービスを使用して、Tanzu CLI によるクライアント接続を提供します。Tanzu CLI は、TKG サービス クラスタのライフサイクルのプロビジョニングと管理をサポートします。各 スーパーバイザー インスタンスは、外部 ID プロバイダを 1 つのみサポートします。

pinniped-auth CLI が機能するように認証プラグインと OIDC 発行者が適切に構成された後、`tanzu login --endpoint` を使用して スーパーバイザー にログインすると、システムはいくつかの既知の構成マップを検索して `pinniped config` 構成ファイルを構築します。

外部 OIDC プロバイダを使用して接続するには、外部 ID プロバイダを使用した スーパーバイザー 上の TKG クラスタへの接続を参照してください。

ハイブリッド アプローチを使用した認証：vCenter SSO と Tanzu CLI

vCenter Single Sign-On を ID プロバイダとして使用しており、Tanzu CLI を使用する場合は、ハイブリッド アプローチを利用し、両方のツールを使用して スーパーバイザー にログインできます。このアプローチは、標準パッケージのインストールに役立つことがあります。Tanzu CLI と vCenter SSO 認証を使用した スーパーバイザーへの接続を参照してください。

DevOps のユーザーとグループ

vSphere 名前空間 の構成時に確立する権限は、DevOps ユーザーが TKG サービス クラスタのライフサイクルを管理するための権限です。権限を割り当てる対象の DevOps ユーザーまたはグループは、ID ソース内にある必要があります。DevOps ユーザーは、ID プロバイダの認証情報を使用して認証します。

DevOps ユーザーは、ダウンストリーム ユーザー（プロビジョニングされたクラスタにワークロードをデプロイする開発者など）にクラスタへのアクセス権を付与します。これらのユーザーは、ID プロバイダまたは Kubernetes のクラスタ ロールとバインドを使用して認証します。詳細については、次のセクションを参照してください。

注： vSphere 名前空間 の権限は、TKG サービス クラスタを作成および管理する必要がある DevOps ユーザー専用です。これらのクラスタのユーザー（開発者など）は Kubernetes 認証メカニズムを使用します。

ロールの権限とバインド

TKGS クラスタには、2 種類のロール ベースのアクセス コントロール (RBAC) システム（vSphere 名前空間 の権限と Kubernetes RBAC 認可）があります。vSphere 管理者は、vSphere 名前空間 の権限を割り当てて、ユーザーが TKG サービス クラスタを作成および操作できるようにします。クラスタ オペレータは、Kubernetes RBAC を使用してクラスタへのアクセス権を付与し、開発者にロールの権限を割り当てます。開発者への TKG サービス クラスタへの vCenter SSO アクセス権付与を参照してください。

vSphere 名前空間 では、編集可能、表示可能、所有者の 3 つのロールがサポートされます。ロールの権限は、TKG サービス クラスタをホストする vSphere 名前空間 で割り当てられ、範囲がその名前空間に設定されます。6 章 TKG サービス クラスタをホストするための vSphere 名前空間 の構成を参照してください。

vSphere 名前空間 で編集可能ロール権限が付与されたユーザー/グループは、その vSphere 名前空間 の TKG サービス クラスタの作成、読み取り、更新、および削除を行うことができます。また、ユーザー/グループを編集可能ロールに割り当てると、システムではその vSphere 名前空間 の各クラスタに RoleBinding を作成し、`cluster-admin` という名前の Kubernetes ClusterRole に権限をバインドします。`cluster-admin` ロールを使用すると、ユーザーはターゲット vSphere 名前空間 で TKG サービス クラスタをプロビジョニングおよび操作できます。このマッピングを表示するには、ターゲットの vSphere 名前空間 から `kubectl get rolebinding` コマンドを使用します。

```
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
```

vSphere 名前空間 で表示可能ロール権限が付与されたユーザー/グループには、その vSphere 名前空間 でプロビジョニングされた TKG サービス クラスタ オブジェクトへの読み取り専用アクセス権が割り当てられます。ただし、編集可能権限とは異なり、表示可能ロールには、その vSphere 名前空間 の TKGS クラスタで作成された Kubernetes RoleBinding がありません。これは、表示可能権限が付与されたユーザー/グループをバインドできる同等の読み取り専用ロールが Kubernetes に存在しないためです。`cluster-admin` 以外のユーザーの場合は、Kubernetes RBAC を使用してアクセス権を付与します。開発者への TKG サービス クラスタへの vCenter SSO アクセス権付与を参照してください。

vSphere 名前空間 で所有者権限が付与されたユーザー/グループは、その vSphere 名前空間 の TKG サービス クラスタを管理できます。また、kubectl を使用して追加の vSphere 名前空間 を作成および削除できます。ユーザー/グループを所有者ロールに割り当てると、システムでは ClusterRoleBinding を作成して ClusterRole にマッピングします。これにより、そのユーザー/グループが kubectl を使用して vSphere 名前空間 を作成および削除できます。同じ方法でこのマッピングを表示することはできません。このマッピングを表示するには、SSH で スーパーバイザー ノードに接続する必要があります。

注： 所有者ロールは、vCenter Single Sign-On ID ソースからのユーザーに対してサポートされます。外部 ID プロバイダからのユーザー/グループで所有者ロールを使用することはできません。

vSphere の権限

次の表に、さまざまな vSphere IaaS control plane 個人設定で必要とされる vSphere 権限のタイプを示します。必要に応じて、ワークロード管理用のカスタム vSphere SSO グループとロールを作成できます。プラットフォーム オペレータ専用のグループおよびロールの作成を参照してください。

表 4-1. vSphere with Tanzu 個人設定の vSphere 権限

個人設定	vSphere のロール	vSphere SSO グループ	vSphere 名前空間
VI/Cloud 管理者	システム管理者	管理者	SSO ユーザー、Active Directory ユーザー
DevOps/プラットフォーム オペレータ	非管理者ロールまたはカスタム ロール	ServiceProviderUsers	SSO ユーザー、Active Directory ユーザー
開発者	読み取り専用、またはなし	なし	SSO ユーザー、Active Directory ユーザー

システム管理者の接続

管理者は、TKG サービス クラスタ ノードに `kubernetes-admin` ユーザーとして接続できます。この方法は、vCenter Single Sign-On 認証が使用できない場合に適していることがあります。トラブルシューティングを行う場合、システム管理者は SSH とプライベート キーを使用することにより、`vmware-system-user` として TKG サービス に接続できます。Kubernetes 管理者およびシステム ユーザーとしての TKG サービス クラスタへの接続を参照してください。

TKG サービス クラスタ用 CLI ツールのインストール

vSphere IaaS control plane では、TKG サービス クラスタのライフサイクルに接続し、それを管理するためのさまざまな CLI ツールが提供されています。

vSphere 向け Kubernetes CLI Tools のインストール

vCenter Single Sign-On ユーザーは、vSphere 向け Kubernetes CLI Tools を使用して TKG サービス クラスタに接続し、操作できます。

vSphere 向け Kubernetes CLI Tools について

vSphere 向け Kubernetes CLI Tools のダウンロード パッケージ (`vsphere-plugin.zip`) には、2 つの実行可能ファイル、kubectl と kubectl 向けの vSphere プラグイン が含まれています。

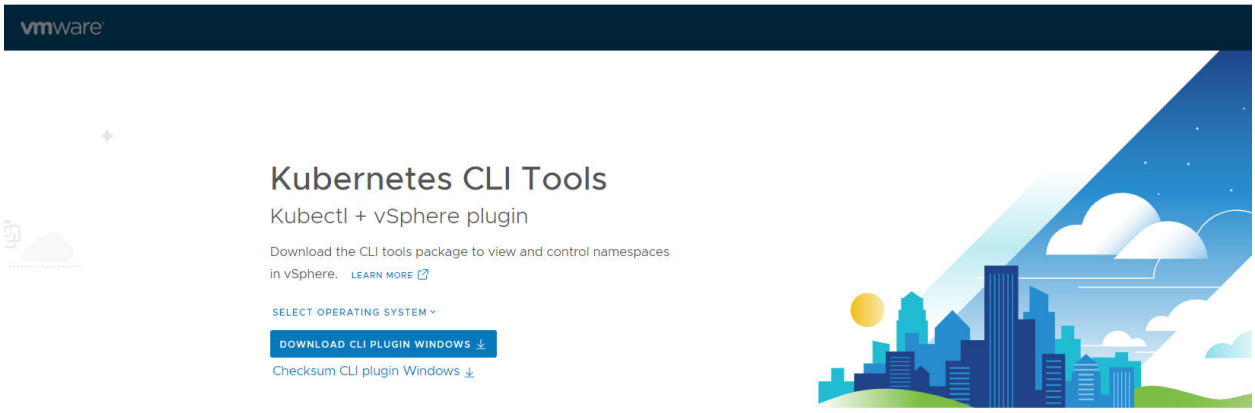
kubectl CLI は、プラグイン可能なアーキテクチャを備えています。kubectl 向けの vSphere プラグイン では、kubectl で使用できるコマンドが拡張されているため、vCenter Single Sign-On を使用して スーパーバイザーおよび TKG クラスタに接続できます。

注： vSphere IaaS control plane は、x86/64 プロセッサ用のバイナリを提供します。

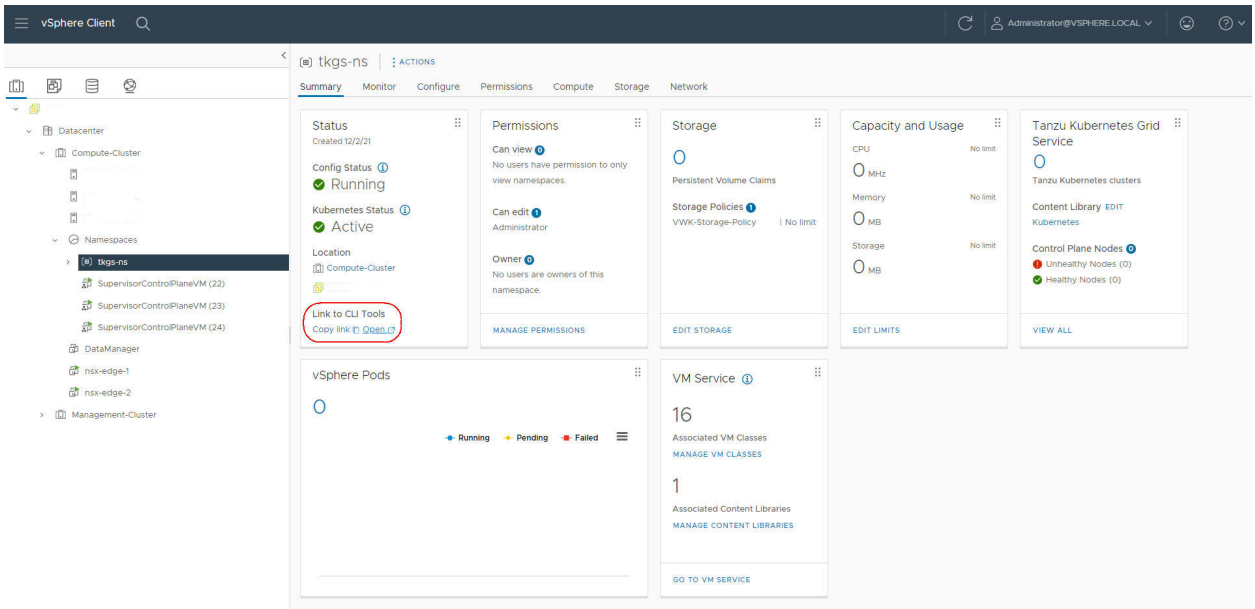
前提条件：vSphere 名前空間 が作成されていること

vSphere 向け Kubernetes CLI Tools は、vSphere IaaS control plane DevOps ページからダウンロードできます。

図 4-1. vSphere DevOps ページ



vSphere IaaS control plane DevOps ページには、vSphere 名前空間 構成パネルからアクセスできます。6 章 TKG サービス クラスタをホストするための vSphere 名前空間 の構成を参照してください。



vSphere Client を使用した vSphere 向け Kubernetes CLI Tools のインストール

vCenter Server の [DevOps] 画面から vSphere 向け Kubernetes CLI Tools をインストールするには、次の手順を実行します。

1 vSphere 管理者から Kubernetes CLI Tools ダウンロード ページのリンクを取得します。または、vCenter Server にアクセスできる場合は、次のようにしてリンクを取得します。

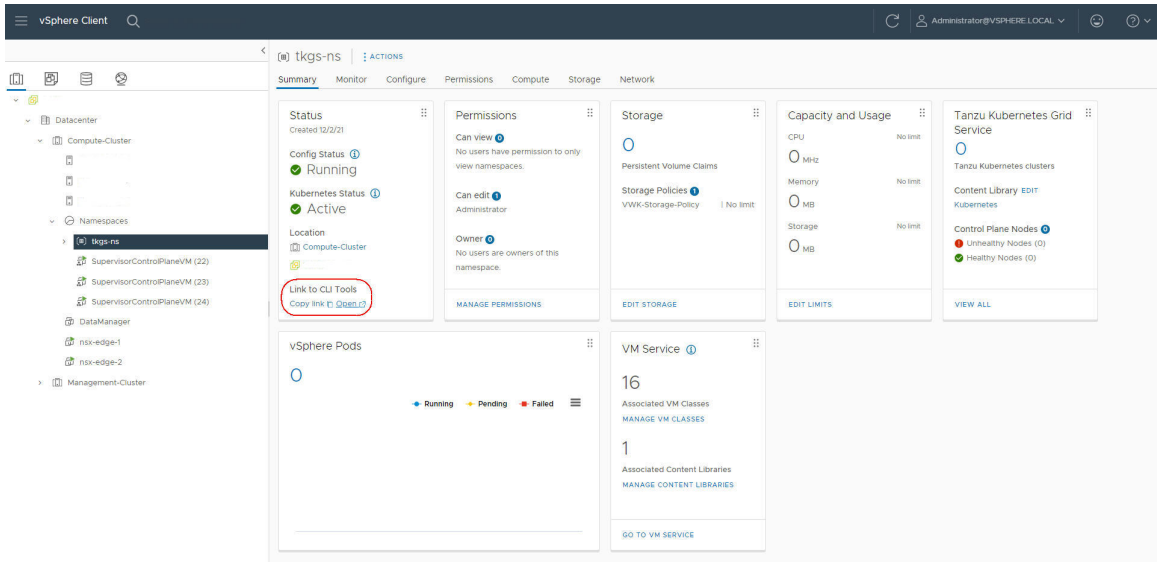
   a vSphere Client を使用して、vCenter Server にログインします。

   b [ワークロード管理] - [名前空間] の順に移動します。

c 作業対象の vSphere 名前空間 を選択します。

d [サマリ] タブを選択し、この画面の [ステータス] 領域を見つけます。(図を参照してください。)

e [CLI ツールへのリンク] という見出しの下にある [開く] を選択してダウンロード ページを開きます。または、リンクを [コピー] することもできます。



2 オペレーティング システムを選択します。

注： 必要に応じて、このトピックの最後にある OS 固有のインストール手順を参照してください。

3 `vsphere-plugin.zip` ファイルをダウンロードします。

4 この ZIP ファイルのコンテンツを作業ディレクトリに解凍します。

5 両方の実行ファイルの場所をシステムの PATH 変数に追加します。

6 kubectl CLI のインストールを確認するには、シェル、ターミナル、またはコマンド プロンプトのセッションを開始し、`kubectl` コマンドを実行します。

kubectl のバナー メッセージと、CLI のコマンドライン オプションのリストが表示されます。

7 kubectl 向けの vSphere プラグイン のインストールを確認するには、`kubectl vsphere` コマンドを実行します。

kubectl 向けの vSphere プラグイン のバナー メッセージと、プラグインのコマンドライン オプションのリストが表示されます。

コマンド ラインからの vSphere 向け Kubernetes CLI Tools のインストール

Linux または MacOS を使用している場合は、次のコマンドを実行して vSphere 向け Kubernetes CLI Tools をダウンロードできます。

1 vSphere Client を使用して、vCenter Server にログインします。

2 [ワークロード管理] - [スーパーバイザー] の順に選択します。

3 スーパーバイザー 制御プレーンのロード バランサの IP アドレスを取得します。

4 これを示す変数を作成します。

5 次のコマンドを実行してツールをインストールします（環境によっては調整が必要です）。

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 `kubectl` コマンドと `kubectl vsphere` コマンドを実行し、インストールを確認します。

Linux への vSphere 向け Kubernetes CLI Tools のインストール

vSphere 向け Kubernetes CLI Tools を Linux ホストにインストールします。

1 Linux 用の `vsphere-plugin.zip` ファイルをダウンロードします。

2 2 つの実行可能ファイル、kubectl と `kubectl-vsphere` であるアーカイブのコンテンツを抽出します。

3 両方の実行可能ファイルを OS の実行可能ファイルの検索パス（`/usr/local/bin` など）に置きます。

4 `kubectl vsphere` コマンドを実行して、インストールを確認します。

5 `kubectl vsphere login --server=Supervisor-IP` コマンド を実行して、スーパーバイザー にログインします。

6 `kubectl config get-contexts` コマンドを実行して、アクセス権のある vSphere 名前空間 のリストを表示します。

7 `kubectl config use-context CONTEXT` コマンドを実行して、デフォルトのコンテキストを選択します。

MacOS への vSphere 向け Kubernetes CLI Tools のインストール

vSphere 向け Kubernetes CLI Tools を MacOS ホストにインストールします。

1 MacOS 用の `vsphere-plugin.zip` ファイルをダウンロードします。

2 2 つの実行可能ファイル、kubectl と `kubectl-vsphere` であるアーカイブのコンテンツを抽出します。

3 両方の実行可能ファイルを OS の実行可能ファイルの検索パス（`/usr/local/bin` など）に置きます。

4 `kubectl vsphere` コマンドを実行して、インストールを確認します。

5 `kubectl vsphere login --server=Supervisor-IP` コマンド を実行して、スーパーバイザー にログインします。

6 `kubectl config get-contexts` コマンドを実行して、アクセス権のある vSphere 名前空間 のリストを表示します。

7 `kubectl config use-context CONTEXT` コマンドを実行して、デフォルトのコンテキストを選択します。

Windows への vSphere 向け Kubernetes CLI Tools のインストール

vSphere 向け Kubernetes CLI Tools を Windows ホストにインストールします。

1 MacOS 用の `vsphere-plugin.zip` ファイルをダウンロードします。

2 2 つの実行可能ファイル、`kubectl.exe` と `kubectl-vsphere.exe` であるアーカイブのコンテンツを抽出します。

3 両方の実行可能ファイルをシステム パスに置きます。

4 `kubectl vsphere` コマンドを実行して、インストールを確認します。

5 `kubectl vsphere login --server=Supervisor-IP` コマンドを実行して、スーパーバイザー にログインします。

6 `kubectl config get-contexts` コマンドを実行して、アクセス権のある vSphere 名前空間 のリストを表示します。

7 `kubectl config use-context CONTEXT` コマンドを実行して、デフォルトのコンテキストを選択します。

TKG サービス クラスタで使用するための Tanzu CLI のインストール

Tanzu CLI を使用して、TKG サービス クラスタのライフサイクルをプロビジョニングおよび管理できます。

TKG サービス クラスタを操作するための Tanzu CLI をインストールします。

手順

1 Tanzu CLI 製品ドキュメントでインストール手順を参照します。

2 互換性マトリックスで適切なバージョンを確認します。

注： TKG サービス 3.0 の場合は、v1.1.0 より新しいバージョンの Tanzu CLI を使用しないでください。たとえば、Tanzu CLI v1.2.0 を使用しないでください。

注： TKG サービス 3.1.0 以降では、TKG サービスのバージョンが同じ pinniped-auth CLI プラグインをダウンロードして、最新バージョンの imgpkg CLI プラグインをダウンロードします。

3 ターゲット バージョンの手順に従って Tanzu CLI をインストールします。

例：VMware Tanzu CLI v1.1.x のインストールおよび使用：https://docs.vmware.com/jp/VMware-Tanzu-CLI/1.1/tanzu-cli/index.html

vSphere Docker 認証情報ヘルパー のインストール

コンテナ イメージをに安全にプッシュし、組み込みの Harbor レジストリからコンテナ イメージをプルするには、vSphere Docker 認証情報ヘルパー CLI を使用します。

vSphere Docker 認証情報ヘルパー を使用して、Docker クライアントをコンテナ レジストリに安全に接続します。

注： vSphere Docker 認証情報ヘルパー は、組み込みの Harbor レジストリで使用することを目的としていますが、これは非推奨です。Harbor スーパーバイザー サービス を使用している場合は、Docker によるそのサービスへの接続方法を「Harbor レジストリ証明書を使用した Docker クライアントの構成」で参照してください。

前提条件

Kubernetes CLI Tools ダウンロード ページには、vSphere Docker 認証情報ヘルパー をダウンロードするためのリンクが掲載されています。

- Docker クライアントを構成します。
- スーパーバイザー で有効になっている組み込みの Harbor レジストリへのアクセス権を取得します。
- vSphere 管理者から vSphere 向け Kubernetes CLI Tools ダウンロード ページのリンクを取得します。
- または、vCenter Server にアクセスできる場合は、次のようにしてリンクを取得します。
  - vSphere Client を使用して、vCenter Server にログインします。
  - [ワークロード管理] - [名前空間] の順に移動して、ターゲットの vSphere 名前空間 を選択します。
  - [サマリ] タブを選択し、[ステータス] ファイルを特定します。
  - [CLI ツールへのリンク] という見出しの下にある [開く] を選択してダウンロード ページを開きます。または、リンクを [コピー] することもできます。

手順

1. ブラウザを使用して、環境に応じた [Kubernetes CLI Tools] ダウンロード URL に移動します。
2. vSphere Docker 認証情報ヘルパー セクションまでスクロールします。
3. オペレーティング システムを選択します。
4. `vsphere-docker-credential-helper.zip` ファイルをダウンロードします。
5. この ZIP ファイルのコンテンツを作業ディレクトリに解凍します。

   `docker-credential-vsphere` バイナリ実行ファイルを使用できます。
6. `docker-credential-vsphere` バイナリを Docker クライアント ホストにコピーします。
7. バイナリの場所をシステム パスに追加します。

   たとえば、Linux では次のようになります。

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

8 シェルまたはターミナル セッションで `docker-credential-vsphere` コマンドを実行して、vSphere Docker 認証情報ヘルパー のインストールを確認します。

バナー メッセージと、CLI のコマンドライン オプションのリストが表示されます。

```
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 レジストリにログインします。

まず、使用量を確認します。

```
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
```

注： vSphere Docker 認証情報ヘルパー では、ユーザー文字列はすべて小文字であることが想定されます。すべてに小文字を使用しない場合、ログインは機能しても、後続の Docker コマンドは機能しないことがあります。ユーザー名のすべての文字には小文字を使用してください。

次のコマンドを使用してログインします。

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

認証トークンが取得され、保存されて、ログイン状態になります。

```
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 Harbor レジストリからログアウトします。

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

vCenter SSO 認証を使用した TKG サービス クラスタへの接続

vCenter Single Sign-On と kubectl 向けの vSphere プラグイン を使用して、またはこれらを Tanzu CLI と組み合わせて、TKG サービス クラスタに接続できます。

vCenter Single Sign-On 認証でのセキュア ログインの構成

スーパーバイザー や TKG サービス クラスタに安全にログインするには、適切な TLS 証明書を使用して kubectl 向けの vSphere プラグイン を構成し、プラグインの最新バージョンが実行されていることを確認します。

スーパーバイザー CA 証明書

vSphere IaaS control plane は、kubectl 向けの vSphere プラグイン コマンド `kubectl vsphere login ...` を使用してクラスタ アクセスのための vCenter Single Sign-On をサポートします。

kubectl 向けの vSphere プラグイン では、デフォルトで安全なログインが行われ、信頼されている証明書が必要とされます。デフォルトは、vCenter Server ルート CA によって署名された証明書です。プラグインは `--insecure-skip-tls-verify` フラグをサポートしていますが、これはセキュリティ上の理由から推奨されません。

kubectl 向けの vSphere プラグイン を使用して スーパーバイザー および TKG サービス クラスタに安全にログインするには、次の 2 つのオプションがあります。

オプション	方法
各クライアント マシンに vCenter Server ルート CA 証明書をダウンロードしてインストールします。	Linux の場合は、「vCenter Server 用の信頼できるルート CA 証明書のダウンロードと Ubuntu クライアントへのインストール」を参照してください。 Windows および Mac の場合は、VMware ナレッジベースの記事「Web ブラウザで証明書に関する警告表示を出さないようにするために vCenter Server のルート証明書をダウンロードしてインストールする方法」を参照してください。
スーパーバイザー で使用される VIP 証明書を、各クライアント マシンが信頼する CA によって署名された証明書に置き換えます。	vSphere IaaS 制御プレーンのインストールと構成を参照してください。

注： vCenter Single Sign-On、vCenter Server 証明書の管理とローテーション、トラブルシューティングなど、vSphere 認証の詳細については、vSphere Authentication のドキュメントを参照してください。

TKG クラスタ CA 証明書

`kubectl` CLI を使用して TKG クラスタ API サーバと安全に接続するには、TKG クラスタ CA 証明書をダウンロードする必要があります。

kubectl 向けの vSphere プラグイン の最新バージョンを使用している場合、TKG クラスタに初めてログインすると、プラグインによって kubeconfig ファイルに TKG クラスタ CA 証明書が登録されます。この証明書は、*`TANZU-KUBERNETES-CLUSTER-NAME`*`-ca` という名前の Kubernetes シークレットにも格納されます。プラグインは、この証明書を使用して、対応するクラスタの認証局データストアの CA 情報をポピュレートします。

スーパーバイザー をアップデートした場合は、プラグインの最新バージョンに更新してください。

vCenter Server 用の信頼できるルート CA 証明書のダウンロードと Ubuntu クライアントへのインストール

この手順に従って vCenter Server 用の信頼できるルート CA 証明書をダウンロードし、Ubuntu クライアントにインストールして、kubectl 向けの vSphere プラグイン を使用して スーパーバイザー と TKG サービス クラスタに安全にログインできるようにします。

1 kubectl 向けの vSphere プラグイン をインストールします。vSphere 向け Kubernetes CLI Tools のインストールを参照してください。

2 [ワークロード管理] が有効になっている vCenter Server 用の信頼できるルート CA 証明書をダウンロードします。

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

3 `download.zip` ファイルのコンテンツを現在のディレクトリに解凍します。

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

4 Linux ディレクトリへのパスを変更します。

```
cd /certs/lin
```

5 `/certs/lin` ディレクトリ内の CA 証明書を一覧表示します (`ls`)。

PEM 形式の 2 つの証明書 (`*.0` と `*.r1`) が表示されます。PEM 形式の証明書は、人間が解読可能な base64 形式の証明書で、`----BEGIN CERTIFICATE----` で始まります。

6 `*.crt` 拡張子を証明書ファイルに追加します。例：

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

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

7 ファイルを OpenSSL 証明書ディレクトリの `/etc/ssl/certs` にコピーします。

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

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

8 スーパーバイザー に安全にログインします。

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

9 TKG サービス クラスタに安全にログインします。

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

vCenter Single Sign-On ユーザーおよびグループに対する vSphere 名前空間 権限の構成

vSphere 名前空間 に権限を設定することにより、そこにプロビジョニングされた TKG 2 クラスタに vCenter Single Sign-On ユーザーとグループがアクセスできるようにします。

vSphere 名前空間 を作成したら、ユーザー/グループを追加してロールを割り当て、TKG 2 クラスタ用に構成します。TKG サービス クラスタ向けの vSphere 名前空間 の構成を参照してください。

前提条件

ユーザー、グループ、およびロールの権限は、vSphere 名前空間 レベルで設定されます。スーパーバイザー および TKG 2 クラスタにアクセスするには、まず vSphere 名前空間 を作成する必要があります。TKG サービス クラスタをホストするための vSphere 名前空間 の作成を参照してください。

手順

1 vSphere Client を使用して、vCenter Server にログインします。

2 [ワークロード管理] - [名前空間] の順に選択します。

3 作成した vSphere 名前空間 を選択します。

4 [権限] - [権限の追加] の順に選択します。

5 [ID ソース] : vCenter SSO のユーザーおよびグループに対して [vsphere.local] を選択します。

注： 外部 ID プロバイダを使用している場合は、外部 ID プロバイダを使用した スーパーバイザー 上の TKG クラスタへの接続を参照してください。

6 [ユーザー/グループの検索] : TKG クラスタ操作または TKG 開発者に対し構成されている vCenter SSO ユーザーまたはグループを選択します。

7 [ロール] : 適切なロール（[表示可能]、[編集可能]、または [所有者]）を選択して、ユーザーまたはグループをロールに割り当てます。

オプション	説明
表示可能	vSphere 名前空間内の TKG クラスタ オブジェクトを読み取ることができます。Kubernetes ロールにマッピングされた権限はありません。ロールの権限とバインドを参照してください。
編集可能	vSphere 名前空間内の TKG クラスタ オブジェクトを作成、読み取り、更新、削除できます。Kubernetes `cluster-admin` として、vSphere 名前空間にプロビジョニングされた TKG クラスタを操作できます。ロールの権限とバインドを参照してください。
所有者	「編集可能」と同じ権限です。kubectl を使用して vSphere 名前空間を作成および管理するための追加の権限があります。vCenter SSO でのみ使用できます。ロールの権限とバインドを参照してください。

8 vSphere 名前空間 の構成を完了します。TKG サービス クラスタ向けの vSphere 名前空間 の構成を参照してください。

Kubectl を使用した vCenter Single Sign-On ユーザーとしての スーパーバイザー への接続

kubectl 向けの vSphere プラグイン を使用して スーパーバイザー に接続し、vCenter Single Sign-On の認証情報を使用して認証します。

スーパーバイザー にログインすると、kubectl 向けの vSphere プラグイン によってクラスタのコンテキストが生成されます。Kubernetes では、設定コンテキストにクラスタ、名前空間、およびユーザーが含まれます。クラスタのコンテキストは `.kube/config` ファイルで確認できます。このファイルは、通常、`kubeconfig` ファイルと呼ばれます。

注： 既存の `kubeconfig` ファイルがある場合は、そのファイルに各クラスタ コンテキストが追加されます。kubectl 向けの vSphere プラグイン は、kubectl 自体が使用する KUBECONFIG 環境変数に従います。必須ではありませんが、`kubectl vsphere login ...` を実行する前にこの変数を設定することで、（情報が現在の `kubeconfig` ファイルに追加されるのではなく）新しいファイルに書き込まれるようにすることができます。

前提条件

- vSphere 向け Kubernetes CLI Tools のインストール。
- vCenter Single Sign-On の認証情報を取得します。
- vSphere 管理者から スーパーバイザー の制御プレーンの IP アドレスを取得します。
- vSphere 管理者から vSphere 名前空間 の名前を取得します。
- vSphere 名前空間 の編集権限があることを確認します。vCenter Single Sign-On ユーザーおよびグループに対する vSphere 名前空間 権限の構成を参照してください。
- 署名を付与する認証局 (CA) を Trust Root としてインストールするか、または証明書を Trust Root として直接追加することにより、Kubernetes 制御プレーンによって提供される証明書がシステムで信頼されることを確認します。vCenter Single Sign-On 認証でのセキュア ログインの構成を参照してください。

手順

1 ログインのコマンド構文とオプションを表示するには、次のコマンドを実行します。

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

2 スーパーバイザー に接続するには、次のコマンドを実行します。

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

FQDN を使用してログインすることもできます。

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

例：

```
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
```

この操作により、Kubernetes API への認証に使用する JSON Web トークン (JWT) を含む設定ファイルが作成されます。

3 認証するには、ユーザーのパスワードを入力します。

スーパーバイザー に接続すると、アクセス可能な設定コンテキストが表示されます。例：

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

4 アクセスする権限のある設定コンテキストの詳細を表示するには、次の `kubectl` コマンドを実行します。

```
kubectl config get-contexts
```

CLI に、使用可能な各コンテキストの詳細が表示されます。

5 コンテキストを切り替えるには、次のコマンドを使用します。

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

Kubectl を使用した vCenter Single Sign-On ユーザーとしての TKG サービス クラスタへの接続

kubectl 向けの vSphere プラグイン を使用して TKG クラスタに接続し、vCenter Single Sign-On の認証情報を使用して認証します。

Tanzu Kubernetes クラスタにログインすると、kubectl 向けの vSphere プラグイン によってクラスタのコンテキストが生成されます。Kubernetes では、設定コンテキストにクラスタ、名前空間、およびユーザーが含まれます。クラスタのコンテキストは `.kube/config` ファイルで確認できます。このファイルは、通常、`kubeconfig` ファイルと呼ばれます。

注： 既存の `kubeconfig` ファイルがある場合は、そのファイルに各クラスタ コンテキストが追加されます。kubectl 向けの vSphere プラグイン は、kubectl 自体が使用する KUBECONFIG 環境変数に従います。必須ではありませんが、`kubectl vsphere login ...` を実行する前にこの変数を設定することで、（情報が現在の `kubeconfig` ファイルに追加されるのではなく）新しいファイルに書き込まれるようにすることができます。

前提条件

vSphere 管理者から次の情報を取得します。

- vCenter Single Sign-On の認証情報を取得します。

- スーパーバイザー 制御プレーンの IP アドレスを取得します。
- vSphere 名前空間 の名前を取得します。
- vSphere 向け Kubernetes CLI Tools のインストール。

手順

1 ログインのコマンド構文とオプションを表示するには、次のコマンドを実行します。

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

2 Tanzu Kubernetes クラスタに接続するには、次のコマンドを実行します。

```
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
```

例：

```
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
```

または、スーパーバイザー が完全修飾ドメイン名 (FQDN) で有効になっている場合は、次のコマンドを実行します。

```
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
```

この操作により、Kubernetes API への認証に使用する JSON Web トークン (JWT) を含む設定ファイルが作成されます。

3 認証するには、vCenter Single Sign-On パスワードを入力します。

操作が成功すると、「`Logged in successfully`」というメッセージが表示され、クラスタに対して kubectl コマンドを実行できます。このコマンドによって `Error from server (Forbidden)` が返された場合、通常、このエラーは必要な権限がないことを意味します。

4 使用可能なコンテキストのリストを取得するには、次のコマンドを実行します。

```
kubectl config get-contexts
```

このコマンドでは、アクセスできる設定コンテキストがリスト表示されます。`tkg-cluster-01` などのターゲット クラスタの設定コンテキストが表示されます。

5 ターゲット クラスタのコンテキストを使用するには、次のコマンドを実行します。

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

6 クラスタ ノードをリスト表示するには、次のコマンドを実行します。

```
kubectl get nodes
```

このクラスタ内の制御プレーンとワーカー ノードが表示されます。

7 すべてのクラスタ ポッドをリスト表示するには、次のコマンドを実行します。

```
kubectl get pods -A
```

アクセス権のあるすべての Kubernetes 名前空間について、このクラスタ内のすべてのポッドが表示されます。ワークロードをデプロイしていない場合は、デフォルトの名前空間にポッドは表示されません。

開発者への TKG サービス クラスタへの vCenter SSO アクセス権付与

開発者ユーザーと開発グループは、TKG サービス クラスタのターゲット ユーザーです。TKG サービス クラスタがプロビジョニングされると、vCenter Single Sign-On 認証を使用して、またはサポートされている外部 ID プロバイダを使用して、開発者にアクセス権を付与できます。

開発者向けの認証

クラスタ管理者は、開発者などの他のユーザーにクラスタへのアクセス権を付与できます。開発者は、ユーザー アカウントを使用して直接、またはサービス アカウントを使用して間接的に、クラスタにポッドをデプロイできます。

- ユーザー アカウント認証の場合、TKG サービス クラスタは vCenter Single Sign-On のユーザーとグループをサポートします。ユーザーまたはグループは、vCenter Server のローカルであるか、サポートされているディレクトリ サーバから同期されます。
- 外部 OIDC のユーザーおよびグループは、vSphere 名前空間 ロールに直接マッピングされます。
- サービス アカウント認証の場合は、サービス トークンを使用できます。詳細については、Kubernetes のドキュメントを参照してください。

クラスタへの開発者ユーザーの追加

開発者にクラスタ アクセスを許可するには：

1 ユーザーまたはグループの Role または ClusterRole を定義し、クラスタに適用します。詳細については、Kubernetes のドキュメントを参照してください。

2 ユーザーまたはグループの RoleBinding または ClusterRoleBinding を作成し、クラスタに適用します。次の例を参照してください。

RoleBinding の例

vCenter Single Sign-On のユーザーまたはグループにアクセス権を付与するには、RoleBinding のサブジェクトに `name` パラメータの値として次のいずれかを含める必要があります。

表 4-2. サポートされているユーザーおよびグループのフィールド

フィールド	説明
`sso:USER-NAME@DOMAIN`	たとえば、`sso:joe@vsphere.local` などのローカル ユーザー名です。
`sso:GROUP-NAME@DOMAIN`	たとえば、`sso:devs@ldap.example.com` などの vCenter Server と統合されたディレクトリ サーバのグループ名です。

次の RoleBinding の例では、Joe という vCenter Single Sign-On ローカル ユーザーが、`edit` というデフォルトの ClusterRole にバインドされます。このロールにより、名前空間（この例では `default` 名前空間）内のほとんどのオブジェクトに対する読み取り/書き込みアクセスが許可されます。

```
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
```

Tanzu CLI と vCenter SSO 認証を使用した スーパーバイザー への接続

次の手順に従って、Tanzu CLI を使用して スーパーバイザー に接続し、vCenter Single Sign-On ユーザーとして認証します。

前提条件

次の前提条件を満たすようにします。

1 vSphere 向け Kubernetes CLI Tools をインストールして構成します。vCenter SSO 認証を使用した TKG サービス クラスタへの接続を参照してください。

2 Tanzu CLI をインストールして初期化します。TKG サービス クラスタで使用するための Tanzu CLI のインストールを参照してください。

Tanzu CLI と vCenter SSO を使用した スーパーバイザー への接続

次の手順を実行します。

1 vCenter SSO ユーザーとして スーパーバイザー に接続します。

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

このアクションにより、Tanzu CLI で使用される スーパーバイザー コンテキストが kubeconfig にポピュレートされます。

2 コンテキストを スーパーバイザー の vSphere 名前空間 に切り替えます。

```
kubectl config get-contexts
```

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

3 Tanzu CLI と vCenter SSO を使用して、スーパーバイザー にログインします。

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

ここで、

- `context_name` は、「supervisor」など、このコンテキストのユーザー定義の名前です。
- `--kubeconfig ~/.kube/config` はローカルの `kubeconfig` ファイルへのパスです。デフォルトは `~/.kube/config` で、`KUBECONFIG` 環境変数によって設定されます。vCenter SSO ユーザー用の スーパーバイザー 構成コンテキストが含まれています。
- `--kubecontext SUPERVISOR-CONTROL-PLANE-IP-ADDRESS` は、`10.179.144.55` などの `SUPERVISOR_IP` と同じ スーパーバイザー のコンテキストです。

4 Tanzu CLI コマンドを実行し、接続を確認します。

```
tanzu plugin list
```

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

外部 ID プロバイダを使用した スーパーバイザー 上の TKG クラスタへの接続

Kubernetes 用の外部 ID プロバイダおよび Pinniped 認証サービスを使用して、スーパーバイザー 上の TKG クラスタに接続できます。

TKG サービス クラスタで使用する外部 ID プロバイダの構成

Okta など、任意の OIDC 準拠 ID プロバイダ (IDP) を使用して スーパーバイザー を構成できます。統合を完了するには、IDP に スーパーバイザー のコールバック URL を構成します。

サポートされている外部 OIDC プロバイダ

任意の OIDC 準拠 ID プロバイダを使用して スーパーバイザー を構成できます。次の表に、一般的なものと、構成手順へのリンクを示します。

外部 ID プロバイダ	構成
Okta	Okta を使用した OIDC 構成の例 Configure Okta as an OIDC provider for Pinniped も参照してください。
Workspace ONE	Configure Workspace ONE Access as an OIDC provider for Pinniped

外部 ID プロバイダ	構成
Dex	Configure Dex as an OIDC provider for Pinniped
GitLab	Configure GitLab as an OIDC provider for Pinniped
Google OAuth	Using Google OAuth 2

スーパーバイザー のコールバック URL を使用した ID プロバイダの構成

スーパーバイザー は、外部 ID プロバイダに対する OAuth 2.0 クライアントとして機能します。スーパーバイザー コールバック URL は、外部 ID プロバイダの構成に使用されるリダイレクト URL です。コールバック URL は、https://SUPERVISOR-VIP/wcp/pinniped/callback という形式になります。

注： ID プロバイダの登録を実行する際、コールバック URL が構成中の OIDC プロバイダで「リダイレクト URL」と呼ばれることがあります。

スーパーバイザー の TKG で使用する外部 ID プロバイダを構成する場合、外部プロバイダに、vCenter Server で利用可能な [コールバック URL] を [ワークロード管理] - [スーパーバイザー] - [構成] - [ID プロバイダ] 画面で入力します。

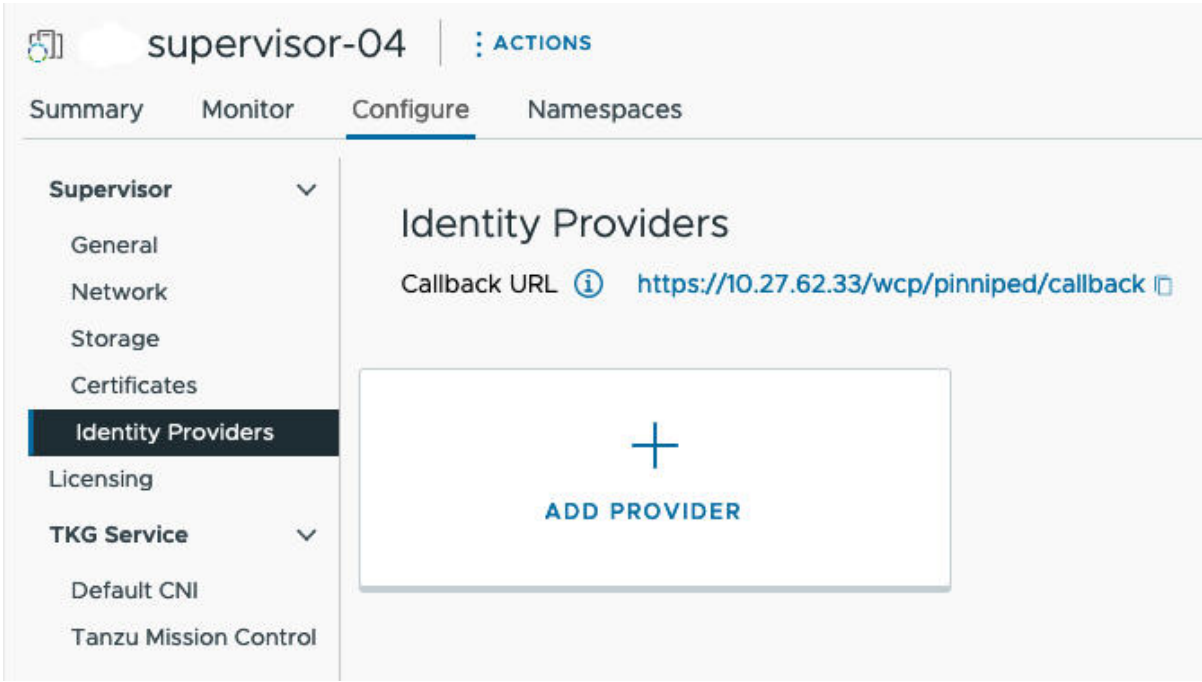
Okta を使用した OIDC 構成の例

Okta では、ユーザーは OpenID Connect プロトコルを使用してアプリケーションにログインできます。スーパーバイザー で Tanzu Kubernetes Grid の外部 ID プロバイダとして Okta を構成する場合、スーパーバイザー と Tanzu Kubernetes Grid クラスタの Pinniped ポッドが vSphere 名前空間 およびワークロード クラスタの両方へのユーザー アクセスを制御します。

1 Okta と vCenter Server 間の OIDC 接続を作成する必要がある ID プロバイダのコールバック URL をコピーします。

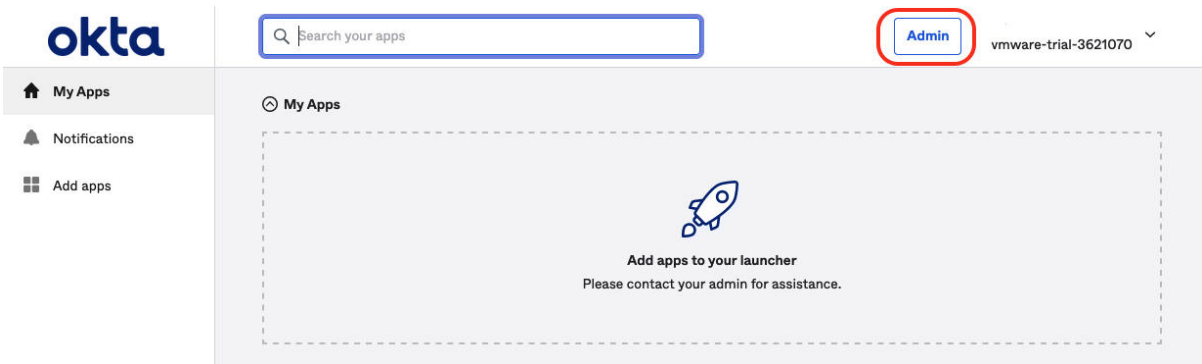
  vSphere Client を使用して、ID プロバイダ コールバック URL を [ワークロード管理] - [スーパーバイザー] - [構成] - [ID プロバイダ] で取得します。この URL を一時的な場所にコピーします。

図 4-2. ID プロバイダ コールバック URL



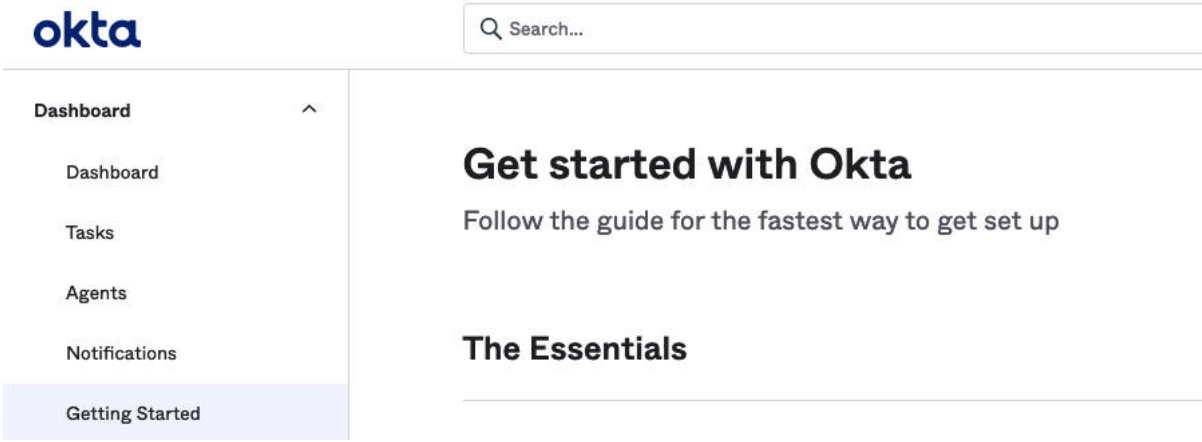
2 組織の Okta アカウントにログインするか、https://www.okta.com/ で評価版アカウントを作成します。[管理] ボタンをクリックして Okta 管理コンソールを開きます。

図 4-3. Okta 管理コンソール



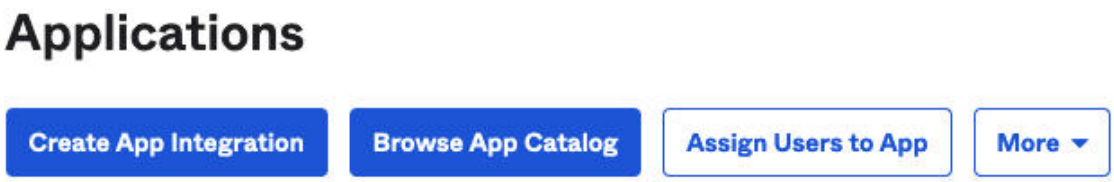
3 管理コンソールの [はじめに] ページから、[アプリケーション] - [アプリケーション] に移動します。

図 4-4. Okta の開始



4 [アプリケーション統合の作成] オプションを選択します。

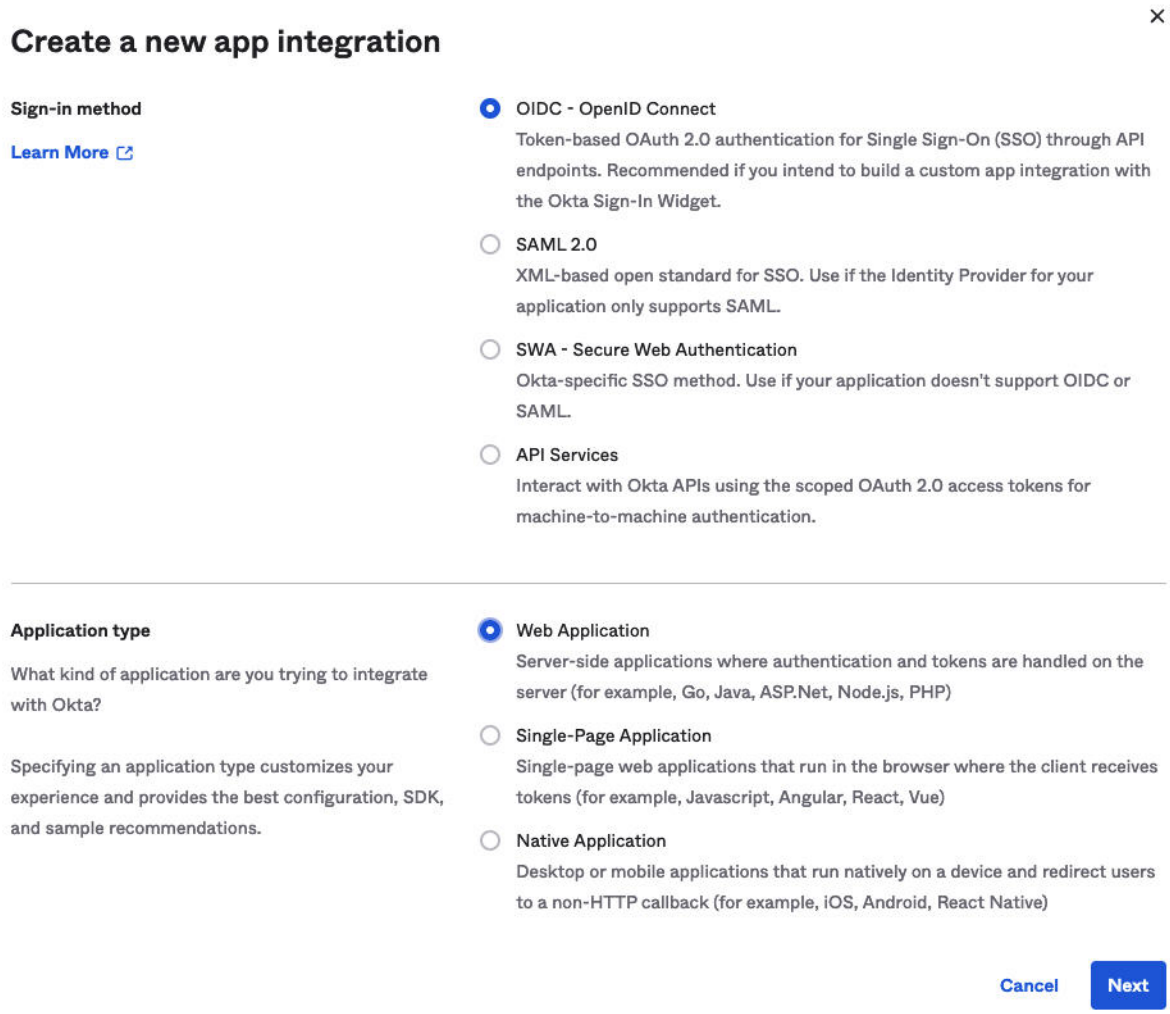
図 4-5. Okta アプリケーション統合の作成



5 新しいアプリケーション統合を作成します。

- ログイン方法を [OIDC - OpenID Connect] に設定します
- アプリケーション タイプを [Web アプリケーション] に設定します

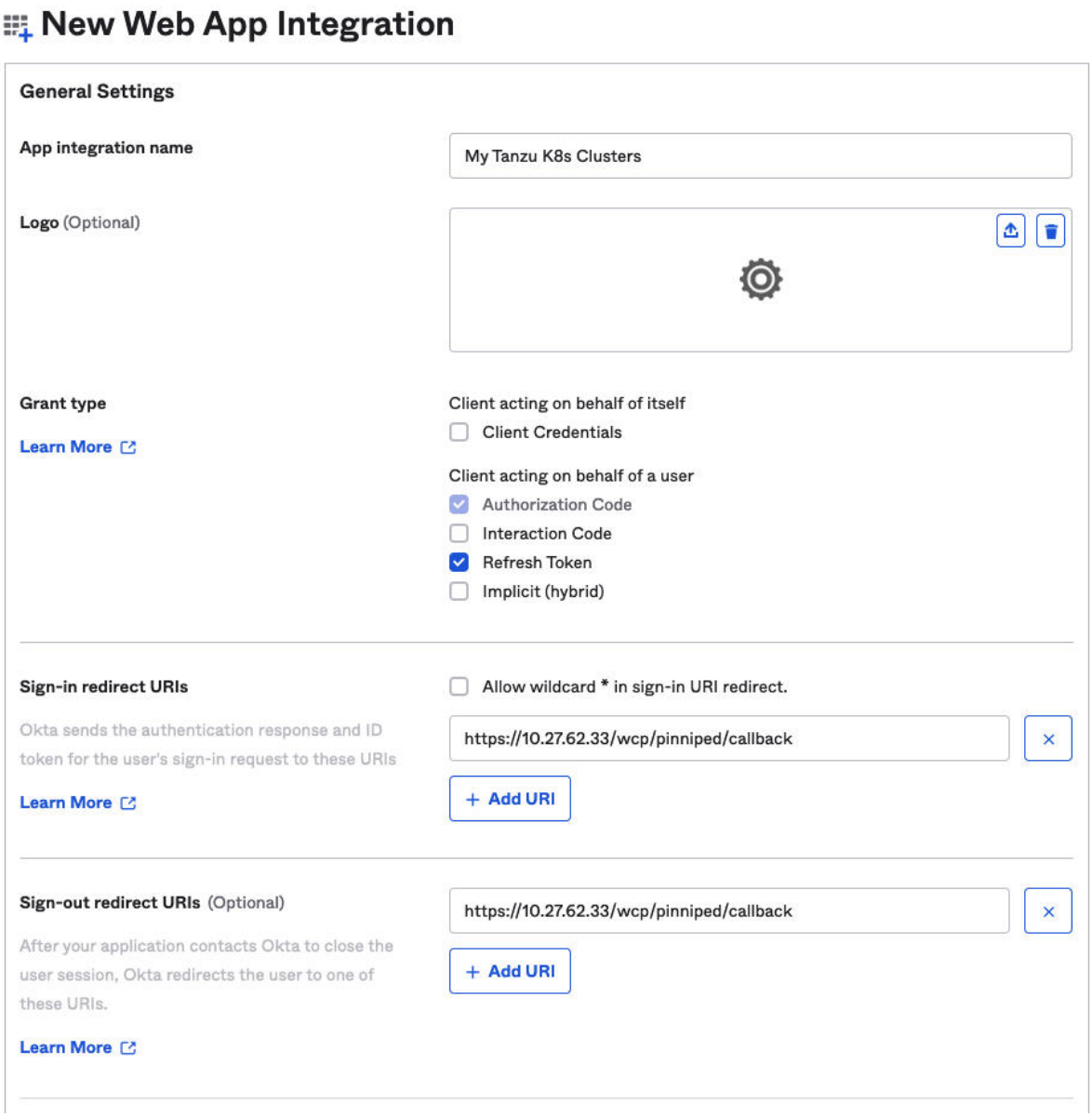
図 4-6. Okta サインオン方法とアプリケーション タイプ



6 Okta の Web アプリケーション統合の詳細を構成します。

- ユーザー定義文字列である [アプリケーション統合名] を指定します。
- [付与タイプ] を指定します：[認可コード] を選択し、[リフレッシュ トークン] も選択します。
- リダイレクト URI にログインします：スーパーバイザー からコピーした（手順 1 を参照）ID プロバイダ コールバック URL（https://10.27.62.33/wcp/pinnipend/callback など）を入力します。
- リダイレクト URI からログアウトします：スーパーバイザー からコピーした（手順 1 を参照）ID プロバイダ コールバック URL（https://10.27.62.33/wcp/pinnipend/callback など）を入力します。

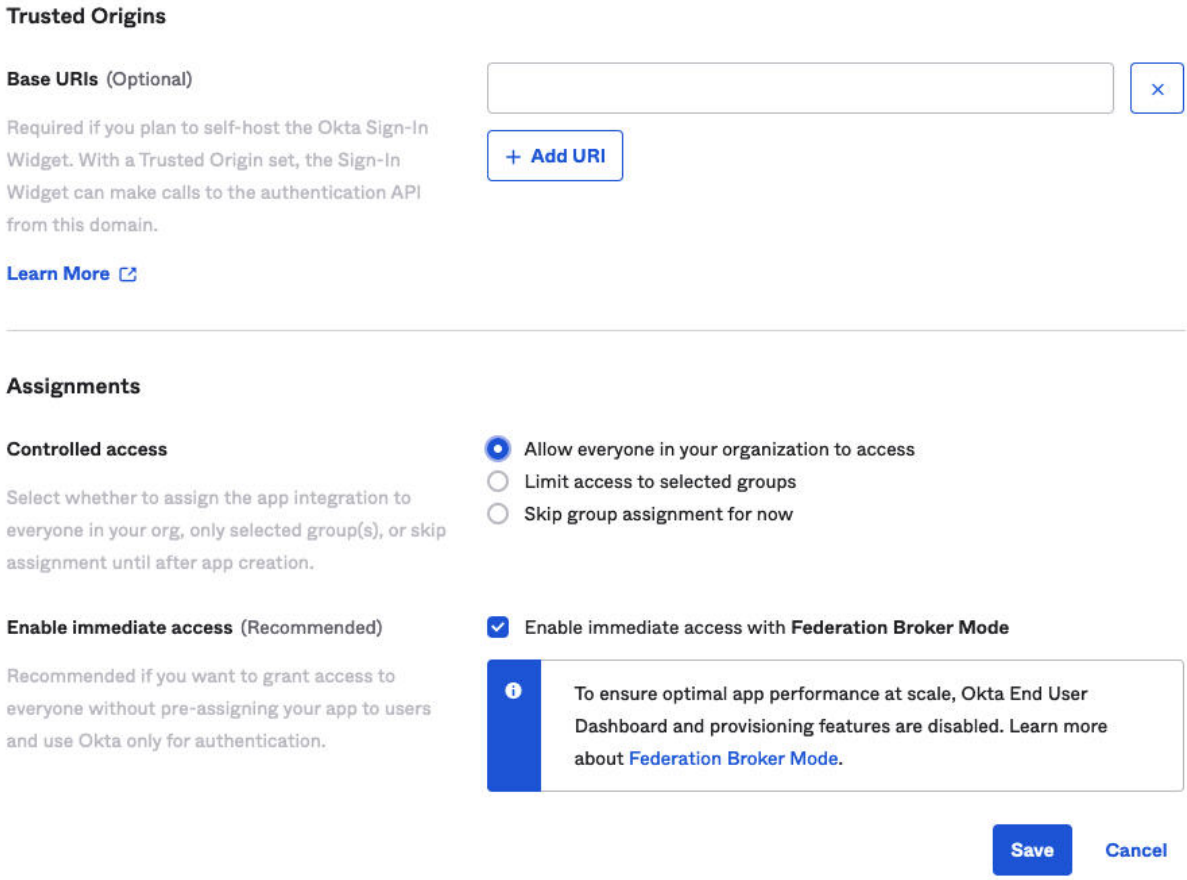
図 4-7. Okta Web アプリケーション統合の詳細



7 ユーザー アクセス コントロールを構成します。

[割り当て] - [制限付きアクセス] セクションで、Tanzu Kubernetes Grid クラスタにアクセス可能な組織に存在する Okta ユーザーを任意で制御できます。例では、組織内で定義されているすべてのユーザーがアクセスできます。

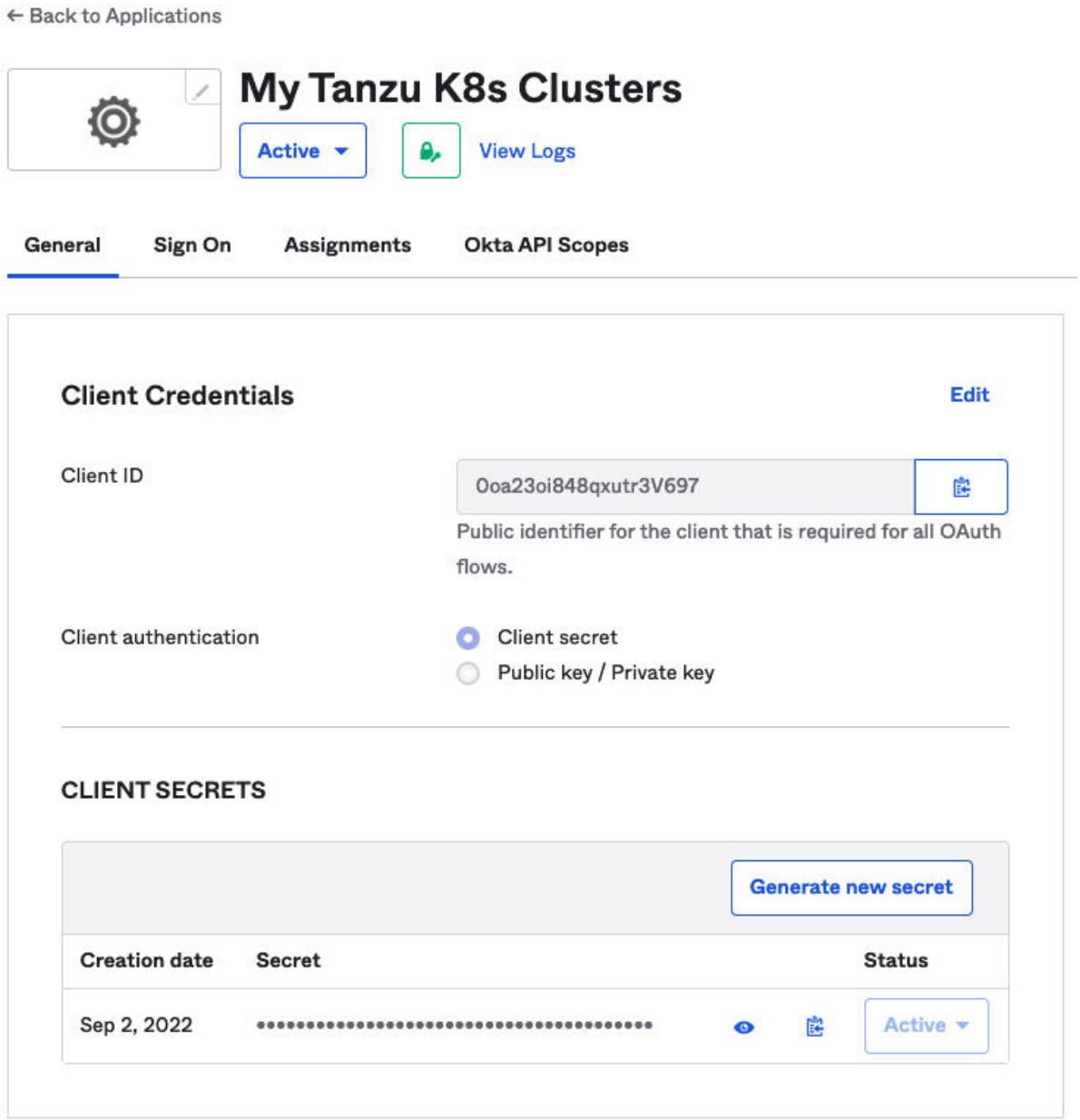
図 4-8. Okta アクセス コントロール



8 [保存] をクリックし、返される [クライアント ID] および [クライアント シークレット] をコピーします。

Okta 構成を保存すると、管理コンソールで [クライアント ID] と [クライアント シークレット] が提供されます。両方のデータをコピーします。これらは スーパーバイザー を外部 ID プロバイダで構成するために必要です。

図 4-9. OIDC クライアント ID およびシークレット

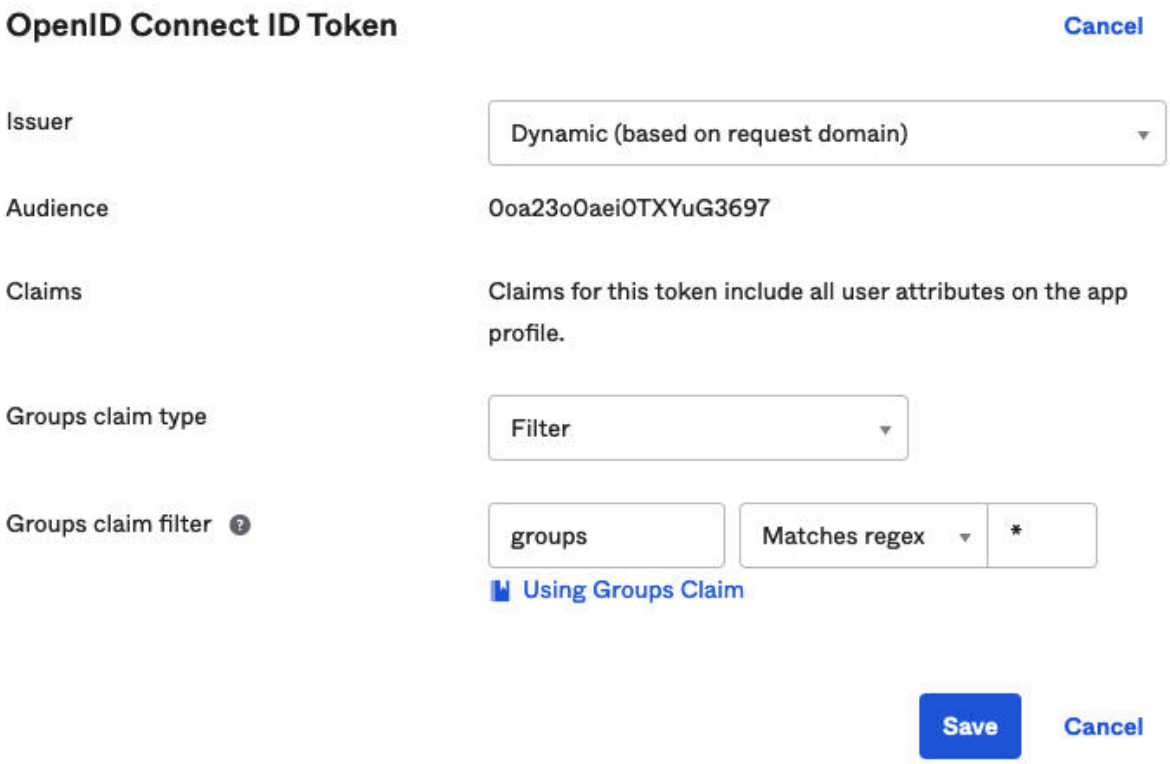


9 OpenID Connect ID トークンを構成します。

[サインオン] タブをクリックします。[OpenID Connect ID トークン] セクションで [編集] リンクをクリックし、[グループ要求タイプ] フィルタを入力し、設定を [保存] します。

たとえば、要求名「グループ」をすべてのグループと一致させるには、[グループ] - [正規表現に一致] - [*] の順に選択します。

図 4-10. OpenID Connect ID トークン

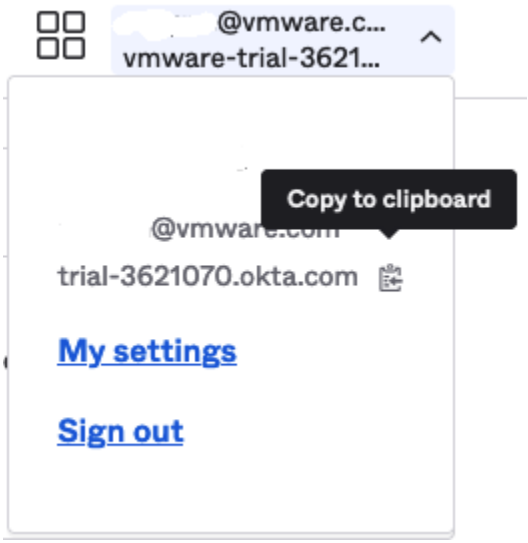


10 [発行者 URL] をコピーします。

スーパーバイザー を構成するには、[クライアント ID] および [クライアント シークレット] に加えて、[発行者 URL] が必要です。

Okta 管理コンソールから [発行者 URL] をコピーします。

図 4-11. Okta 発行者 URL



スーパーバイザー への外部 IDP の登録

Tanzu CLI を使用して スーパーバイザー 上の Tanzu Kubernetes Grid 2.0 クラスタに接続するには、OIDC プロバイダを スーパーバイザー に登録します。

前提条件

外部 ODIC プロバイダを スーパーバイザー に登録する前に、次の前提条件を満たしている必要があります。

- ワークロード管理を有効にして、スーパーバイザー インスタンスをデプロイします。「スーパーバイザーでの TKG 2.0 クラスタの実行」を参照してください。
- 外部 OpenID Connect ID プロバイダを スーパーバイザー コールバック URL で構成します。TKG サービス クラスタで使用する外部 ID プロバイダの構成を参照してください。
- クライアント ID、クライアント シークレット、および発行者 URL を外部 IDP から取得します。TKG サービス クラスタで使用する外部 ID プロバイダの構成を参照してください。

スーパーバイザー への外部 IDP の登録

スーパーバイザー は、ポッドとして Pinniped Supervisor コンポーネントと Pinniped Concierge コンポーネントを実行します。Tanzu Kubernetes Grid クラスタは、ポッドとして Pinniped Concierge コンポーネントのみを実行します。これらのコンポーネントとその相互作用の詳細については、Pinniped 認証サービスのドキュメントを参照してください。

外部 ID プロバイダを スーパーバイザー に登録すると、スーパーバイザー 上の Pinniped Supervisor ポッドと Pinniped Concierge ポッド、Tanzu Kubernetes Grid クラスタ上の Pinniped Concierge ポッドが更新されます。その Tanzu Kubernetes Grid インスタンスで実行されているすべての スーパーバイザー クラスタは自動的に同じ外部 ID プロバイダで構成されます。

外部 ODIC プロバイダを スーパーバイザー に登録するには、次の手順を実行します。

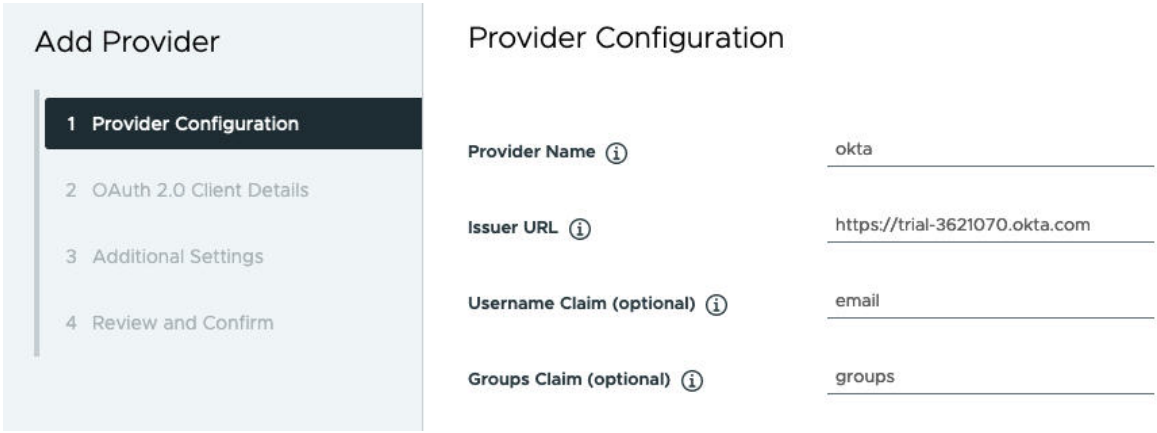
1 vSphere Client を使用して、vCenter Server にログインします。

2 [ワークロード管理] - [スーパーバイザー] - [構成] - [ID プロバイダ] の順に選択します。

3 プラス記号をクリックして登録プロセスを開始します。

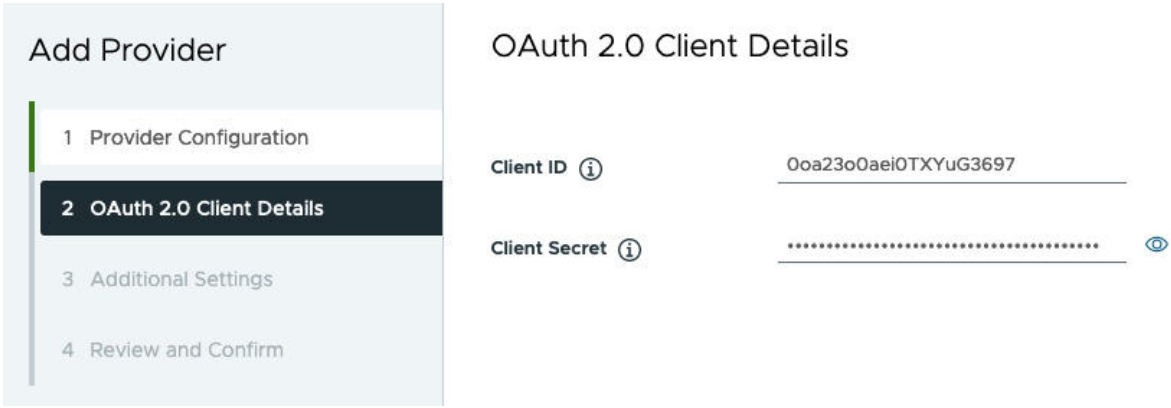
4 プロバイダを構成します。OIDC プロバイダの構成を参照してください。

図 4-12. OIDC プロバイダの構成



5 OAuth 2.0 クライアントの詳細を構成します。OAuth 2.0 クライアントの詳細を参照してください。

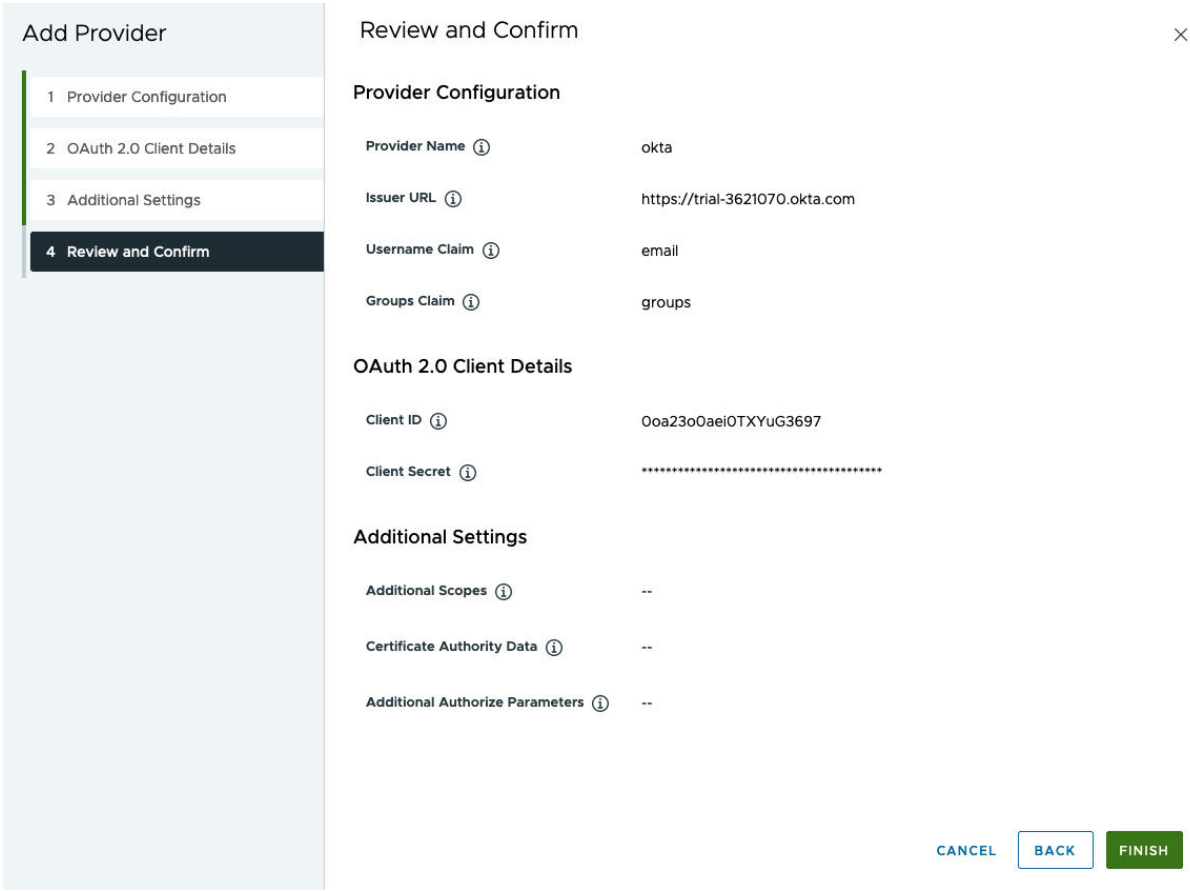
図 4-13. OAuth 2.0 クライアントの詳細



6 追加の設定を構成します。その他の設定を参照してください。

7 プロバイダ設定を確認します。

図 4-14. プロバイダ設定の確認



8 [終了] をクリックして、OIDC プロバイダの登録を完了します。

OIDC プロバイダの構成

外部 OIDC プロバイダを スーパーバイザー に登録する場合は、次のプロバイダ構成の詳細を参照してください。

表 4-3. OIDC プロバイダの構成

フィールド	重要度	説明
プロバイダ名	必須	外部 ID プロバイダのユーザー定義名。
発行者 URL	必須	トークンを発行する ID プロバイダの URL。OIDC 検出 URL は、発行者 URL から取得されます。 たとえば、Okta の発行者 URL は https://trial-4359939-admin.okta.com のように表示され、管理コンソールから取得できます。

表 4-3. OIDC プロバイダの構成 （続き）

フィールド	重要度	説明
ユーザー名の要求	オプション	指定されたユーザーのユーザー名を取得する際に検査を行う、アップストリーム ID プロバイダの ID トークンまたはユーザー情報エンドポイントからの要求。このフィールドを空のままにすると、アップストリーム発行者の URL が「サブ」要求と連結され、Kubernetes で使用されるユーザー名が生成されます。 このフィールドは、認証の判断を行うために、Pinniped がアップストリーム ID トークンの何を確認する必要があるかを指定します。指定しない場合、ユーザー ID は https://IDP-ISSUER?sub=UUID の形式になります。
グループの要求	オプション	指定されたユーザーのグループを取得する際に検査を行う、アップストリーム ID プロバイダの ID トークンまたはユーザー情報エンドポイントからの要求。このフィールドを空のままにすると、アップストリーム ID プロバイダのグループは使用されません。 [グループの要求] フィールドでは、ユーザー ID を認証するためにアップストリーム ID トークンから確認する内容が Pinniped に指示されます。

OAuth 2.0 クライアントの詳細

外部 OIDC プロバイダを スーパーバイザー に登録する場合は、次のプロバイダ OAuth 2.0 クライアントの詳細を参照してください。

表 4-4. OAuth 2.0 クライアントの詳細

OAuth 2.0 クライアントの詳細	重要度	説明
クライアント ID	必須	外部 IDP のクライアント ID
クライアント シークレット	必須	外部 IDP のクライアント シークレット

その他の設定

外部 OIDC プロバイダを スーパーバイザー に登録する場合は、次の追加設定を参照してください。

表 4-5. その他の設定

設定	重要度	説明
追加の範囲	オプション	トークンで要求される追加の範囲
認証局データ	オプション	セキュアな外部 IDP 接続のための TLS 証明局データ
追加の認証パラメータ	オプション	OAuth2 認証要求中の追加パラメータ

外部 ID プロバイダのユーザーおよびグループに対する vSphere 名前空間 権限の構成

OIDC ユーザーの TKG 2.0 クラスタ アクセスを構成するには、外部 ID プロバイダのユーザーおよびグループに対し vSphere 名前空間 を構成します。

外部 ID プロバイダのユーザーおよびグループに対する vSphere 名前空間 権限の構成

スーパーバイザー の TKG 2.0 クラスタは、vSphere 名前空間 にプロビジョニングされます。外部 OIDC プロバイダを スーパーバイザー に登録したら、外部 OIDC プロバイダのユーザーおよびグループのロール権限を使用して vSphere 名前空間 を構成します。このアクションにより、その vSphere 名前空間 の各 TKG 2.0 クラスタで外部 OIDC プロバイダに対しロール バインドが作成されます。既存の vSphere 名前空間 がある場合は、ロール バインドが更新されます。

注： 外部 ID プロバイダを スーパーバイザー に登録すると、その スーパーバイザー で作成されたすべての TKG 2.0 クラスタが、Pinniped コンポーネントを介して外部 ID プロバイダで自動的に構成されます。

1 外部 ID プロバイダを スーパーバイザー に登録します。

スーパーバイザー への外部 IDP の登録を参照してください。

2 1 つ以上の TKG クラスタについて、vSphere 名前空間 を作成するか既存の vSphere 名前空間 を選択します。

TKG サービス クラスタをホストするための vSphere 名前空間 の作成を参照してください。

3 vSphere 名前空間 のユーザーとロールを構成します。

ID ソースとして外部 OIDC プロバイダを選択し、ユーザーを追加してロールを割り当てます。

a [vSphere 名前空間] を選択します。

b [権限] - [権限の追加] の順に選択します。

c [ID ソース] : スーパーバイザー に登録した外部 ID プロバイダを選択します。

外部 ID プロバイダの登録に使用した [プロバイダ名] がドロップダウン メニューに表示されます。表示されない場合は、構成を確認します。

d [ユーザー/グループ検索] : ユーザー名またはグループ名を入力します。テキスト入力は自由形式の文字列です。

外部 ID プロバイダのユーザーとグループは vCenter Server と同期されていないため、選択できません。文字列値（通常はメール アドレス）を入力する必要があります。プリフィックスがないため、たとえば「jdoe@acme.com」と入力できます。

e [ロール] : ロールとして [表示可能] または [編集可能] を選択して、ユーザーまたはグループをロールに割り当てます。

注： 所有者ロールは、外部 ID プロバイダでは使用できません。

4 vSphere 名前空間 の構成を完了します。

TKG サービス クラスタ向けの vSphere 名前空間 の構成を参照してください。

Tanzu CLI と外部 ID プロバイダを使用した スーパーバイザー への接続

次の手順に従い、Tanzu CLI を使用して スーパーバイザー に接続します。

前提条件

次の前提条件を満たすようにします。

1 OIDC に準拠する外部 ID プロバイダを スーパーバイザー に登録します。スーパーバイザー への外部 IDP の登録を参照してください。

2 vSphere 名前空間 へのアクセス権を OIDC ユーザーおよびグループに付与します。外部 ID プロバイダのユーザーおよびグループに対する vSphere 名前空間 権限の構成を参照してください。

Tanzu CLI を使用した スーパーバイザー への接続

次の手順を実行します。

注： TKG サービス 3.1 以降を使用している場合は、TKG サービスと同じバージョンの pinniped-auth cli プラグインと、最新バージョンの imgpkg cli プラグインをダウンロードします。詳細については、Tanzu CLI 製品ドキュメントを参照してください。

1 Tanzu CLI をインストールして初期化します。TKG サービス クラスタで使用するための Tanzu CLI のインストールを参照してください。

2 次のコマンドを実行して、スーパーバイザー に接続します。

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

ここで、

- `context_name` 値は、アクセスが付与されている OIDC の名前です
- `--endpoint` 値は、スーパーバイザー 制御プレーンの IP アドレスです

注： トラブルシューティングを行うには、`--stderr-only` をコマンドに追加します（例：`tanzu login --endpoint https://IP --name USER --stderr-only`）。

3 チャレンジが発行されたら、ブラウザを使用してリンクにアクセスします。

図 4-15. Tanzu CLI ログイン

4 認証コードをコピーして Tanzu CLI に貼り付けます。

```
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 認証が完了すると、Tanzu CLI を使用して、TKG クラスタをアクセス権のあるターゲットの vSphere 名前空間 にプロビジョニングできます。Tanzu CLI を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。

Tanzu CLI を使用した OIDC ユーザーとしての TKG クラスタへの接続

Tanzu CLI を使用して TKG クラスタに接続し、OIDC プロバイダを使用して認証します。

前提条件

これらの手順では、サポートされている外部 ID プロバイダ (IDP) を使用して スーパーバイザー が構成されていること、ユーザー（DevOps ユーザー）が Tanzu CLI を使用して スーパーバイザー に接続していること、および TKG クラスタをプロビジョニング済みであることを前提としています。必要に応じて、次のトピックを参照してください。

- 外部 ID プロバイダを使用した スーパーバイザー 上の TKG クラスタへの接続
- 7 章 TKG サービス クラスタのプロビジョニング

DevOps ユーザーのワークフロー

ターゲット vSphere 名前空間 に対する編集権限を持つ DevOps ユーザーは、Tanzu CLI を使用して共有可能な `kubeconfig` ファイルを生成し、TKG クラスタ ユーザーに配布します。Kubernetes では、設定コンテキストにクラスタ、名前空間、およびユーザーが含まれます。クラスタのコンテキストは `.kube/config` ファイルで確認できます。このファイルは、通常、`kubeconfig` ファイルと呼ばれます。

注： これらの手順は、ターゲット vSphere 名前空間 に対する編集権限を持つ DevOps ユーザーが実行する必要があります。

1 Tanzu CLI コンテキストが スーパーバイザー に設定されていることを確認します。

Tanzu CLI と外部 ID プロバイダを使用した スーパーバイザー への接続を参照してください。

2 vSphere 管理者がターゲット vSphere 名前空間 に対するユーザー権限を構成済みであることを確認します。

外部 OIDC のユーザーおよびグループは、vSphere 名前空間 ロールに直接マッピングされます。共有可能な kubeconfig を生成する前に、クラスタ ユーザーを vSphere 名前空間 に追加する必要があります。

外部 ID プロバイダのユーザーおよびグループに対する vSphere 名前空間 権限の構成を参照してください。

3 ターゲット vSphere 名前空間 にプロビジョニングされた TKG クラスタを一覧表示します。

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

4 ターゲット TKG クラスタ用の共有可能な kubeconfig ファイルを生成します。

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

5 共有の kubeconfig ファイルをクラスタ ユーザーに配布して、そのユーザーが TKG クラスタにログインできるようにします。

クラスタ ユーザーのワークフロー

次の手順を実行して、TKG クラスタにクラスタ ユーザーとしてログインします。

1 DevOps ユーザーから kubeconfig ファイルを取得します。

2 kubeconfig ファイルと kubectl を使用して TKGS クラスタにログインします。

```
kubectl --kubeconfig
```

3 ブラウザ認証プロセスを完了します。

a チャレンジが発行されたら、ブラウザを使用してリンクにアクセスします。

b 認証コードをコピーして CLI に貼り付けます。

4 `kubectl` を使用してクラスタを操作します。

Kubernetes 管理者およびシステム ユーザーとしての TKG サービス クラスタへの接続

Kubernetes 管理者およびシステム ユーザーとして TKG サービス クラスタに接続し、TKG サービス クラスタを管理およびトラブルシューティングできます。

Kubernetes 管理者として TKG サービス クラスタ制御プレーンに接続する

`kubernetes-admin` ユーザーとして TKG サービス クラスタ制御プレーンに接続して、管理者のタスクおよびクラスタの問題のトラブルシューティングを行うことができます。

プロビジョニングされた Tanzu Kubernetes クラスタで有効な `kubeconfig` ファイルは、*`TKG-CLUSTER-NAME`*`-kubeconfig` という名前のシークレット オブジェクトとして スーパーバイザー から入手できます。このシークレットを使用することで、クラスタ制御プレーンに `kubernetes-admin` ユーザーとして接続できます。

手順

1 スーパーバイザー に接続します。

2 ターゲット TKG クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

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

3 名前空間内のシークレット オブジェクトを表示します。

```
kubectl get secrets
```

シークレットには、*TKG-CLUSTER-NAME*-kubeconfig という名前が付いています。

```
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 次のコマンドを実行してシークレットをデコードします。

シークレットは Base64 でエンコードされています。デコードするには、Linux の場合は `base64 --decode`（または `base64 -d`）、MacOS の場合は `base64 --Decode`（または `base64 -D`）、Windows の場合はオンライン ツールを使用します。

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

このコマンドはシークレットをデコードし、`tkgs-cluster-kubeconfig-admin` という名前のローカルファイルに書き込みます。`cat` コマンドを使用して、ファイルの内容を確認します。

5 デコードされた `tkg-cluster-kubeconfig-admin` ファイルを使用し、Kuberentes 管理者として TKG クラスタに接続します。

これには、以下の 2 つの方法があります。

オプション	説明
--kubeconfig <path\to\kubeconfig>	`--kubeconfig` フラグとローカルの kubeconfig ファイルへのパスを使用します。たとえば、kubeconfig ファイルが同じディレクトリにある場合は次のコマンドを実行します：`kubectl --kubeconfig tkg-cluster-kubeconfig-admin get nodes`
KUBECONFIG	デコードされた kubeconfig ファイルを参照するように KUBECONFIG 環境変数を設定し、`kubectl get nodes` などの kubectl を実行します。

クラスタ内にノードが表示されます。

6 vSphere 名前空間 に対する編集権限を持つ DevOps ユーザーが Tanzu CLI を使用して管理者ユーザーとして TKG クラスタにログインする場合は、次のコマンドを実行します。

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

このコマンドは、kubernetes-control-plane の証明書/プライベート キーを含む `kubeconfig` を生成します（すべての認可がバイパスされます）。続いて、この `kubeconfig` を使用してクラスタにログインできます。#unique_36 を参照してください。

プライベート キーを使用した、システム ユーザーとしての TKG サービス クラスタ ノードへの SSH 接続

プライベート キーを使用して、`vmware-system-user` として TKG クラスタ ノードに SSH で接続できます。

`vmware-system-user` ユーザーとして任意の TKG クラスタ ノードに SSH で接続できます。SSH プライベート キーを含むシークレットには、`CLUSTER-NAME-ssh` という名前が付いています。Kubectl を使用した TKG クラスタ シークレットの取得を参照してください。

プライベート キーを使用して SSH 経由で TKG クラスタ ノードに接続するには、スーパーバイザー にジャンプ ボックス vSphere ポッド を作成します。

前提条件

このタスクでは、SSH 接続のジャンプ ホストとして vSphere ポッド をプロビジョニングします。vSphere ポッド を使用するには、スーパーバイザー の NSX ネットワークが必要です。スーパーバイザー に Distributed Switch ネットワークを使用している場合は、パスワードを使用した、システム ユーザーとしての TKG サービス クラスタ ノードへの SSH 接続を参照してください。

手順

1 スーパーバイザー に接続します。

Kubectl を使用した vCenter Single Sign-On ユーザーとしての スーパーバイザー への接続を参照してください。

2 **`NAMESPACE`** という名前の環境変数を作成し、値に、ターゲット TKG クラスタがプロビジョニングされている vSphere 名前空間 の名前を指定します。

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

3 Tanzu Kubernetes クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

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

4 *`TKG-CLUSTER-NAME`*`-ssh` シークレット オブジェクトを表示します。

```
kubectl get secrets
```

5 Docker Hub レジストリ認証情報シークレットを作成します。

デフォルトでは、vSphere Pod (PhotonOS) の作成に使用するイメージは Docker Hub からプルされます。イメージを正常にプルするには、認証情報シークレットが必要になる場合があります。プライベート レジストリ認証情報シークレットの作成を参照してください。

6 次の jumpbox.yaml を使用して、vSphere ポッド を作成します。

namespace 値の YOUR-NAMESPACE を、ターゲット クラスタがプロビジョニングされている vSphere 名前空間 に置き換えます。secretName 値の YOUR-CLUSTER-NAME-ssh をターゲット クラスタの名前に置き換えます。

```
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 jumpbox.yaml の仕様を適用して、ポッドをデプロイします。

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

```
pod/jumpbox created
```

8 ポッドが実行されていることを確認します。

```
kubectl get pods
```

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

注： vSphere 名前空間 内の vCenter Server のジャンプボックス ポッドも確認する必要があります。

9 次の一連のコマンドを実行して、ターゲット クラスタ ノードの IP アドレスを持つ環境変数を作成します。

a ターゲット仮想マシンの名前を取得します。

```
kubectl get virtualmachines
```

b 値がターゲット ノードの名前である `VMNAME` 環境変数を作成します。

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

c 値がターゲット ノード仮想マシンの IP アドレスである `VMIP` 環境変数を作成します。

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

10 次のコマンドを実行して、ジャンプ ボックス ポッドを使用してクラスタ ノードに SSH 接続します。

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

重要： コンテナを作成してソフトウェアをインストールするには、約 60 秒かかります。「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」というエラーが表示された場合は、数秒後に再実行してください。

11 **yes** を入力して、ホストの信頼性を確認します。

```
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 ターゲット ノードに `vmware-system-user` としてログインしていることを確認します。

たとえば、次の出力は、システム ユーザーとして制御プレーン ノードにログインしていることを示しています。

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

13 ノードで必要な操作を実行します。

注目： kubelet の再起動などの特定の処理をノードで実行するには、`sudo` または `sudo su` を使用しなければならない場合があります。

14 操作完了後、**exit** を入力し、vSphere ポッド の SSH セッションからログアウトします。

15 ポッドを削除するには、`kubectl delete pod jumpbox` コマンドを実行します。

注意： セキュリティを確保するには、作業の完了後にジャンプボックス ポッドを削除することを検討してください。必要に応じて、後でポッドを再作成できます。

パスワードを使用した、システム ユーザーとしての TKG サービス クラスタ ノードへの SSH 接続

パスワードを使用して、`vmware-system-user` としてワークロード クラスタ ノードに SSH 接続できます。

パスワードを使用して、`vmware-system-user` ユーザーとしてクラスタ ノードに接続できます。パスワードは、`CLUSTER-NAME-ssh-password` という名前のシークレットとして保存されます。パスワードは `.data.ssh-passwordkey` に base64 でエンコードされています。SSH セッションを介してパスワードを指定できます。Kubectl を使用した TKG クラスタ シークレットの取得を参照してください。

前提条件

SSH 接続を適切なワークロード ネットワークにルーティングするには、[ワークロード管理] が有効な vSphere 環境に Linux ジャンプ ホスト仮想マシンをデプロイします。Linux ジャンプ ホスト仮想マシンの作成を参照してください。

注： vDS ネットワークを使用しており、SSH を使用してクラスタ ノードに接続する場合は、ジャンプ ホスト仮想マシンをデプロイすることが困難な場合があります。プライベート キーの代わりにパスワードを使用して SSH 接続すると、NSX ネットワークでもこの方法を使用できます。

手順

1 ジャンプ ホスト仮想マシンの IP アドレス、ユーザー名、およびパスワードを取得します。

Linux ジャンプ ホスト仮想マシンの作成を参照してください。

2 スーパーバイザー に接続します。

Kubectl を使用した vCenter Single Sign-On ユーザーとしての スーパーバイザー への接続を参照してください。

3 ターゲット TKG クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

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

4 ターゲット クラスタ ノードの IP アドレスを取得します。

ノードを一覧表示します。

```
kubectl get virtualmachines
```

ターゲット ノードの IP アドレスを取得するノードを記述します。

```
kubectl describe virtualmachines
```

5 `TKG-CLUSTER-NAME-ssh-password` シークレットを表示します。

```
kubectl get secrets
```

6 ターゲット クラスタの ssh-passwordkey を取得します。

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

たとえば、ssh-passwordkey が返されます。

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

7 ssh-passwordkey をデコードします。

シークレットは Base64 でエンコードされています。デコードするには、Linux の場合は `base64 --decode`（または `base64 -d`）、MacOS の場合は `base64 --Decode`（または `base64 -D`）、Windows の場合はオンライン ツールを使用します。

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

8 ターゲット クラスタ ノードに `vmware-system-user` として SSH 接続します。

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

9 デコードしたパスワードを使用してログインします。

Linux ジャンプ ホスト仮想マシンの作成

パスワードを使用してワークロード クラスタ ノードに SSH 接続するには、SSH トンネル接続のためにワークロード ネットワークと管理またはフロントエンド ネットワークに接続するジャンプ ボックス仮想マシンを作成します。

Linux ジャンプ ホスト仮想マシンの作成

Linux ジャンプ ボックス仮想マシンを作成するには、次の手順に従います。これを実現するには、さまざまな方法があります。ここに示すのは、そのうちの 1 つです。この手順では、https://github.com/vmware/photon/wiki/Downloading-Photon-OS からダウンロードできる PhotonOS を使用します。

注： ジャンプ ホストを作成するこの方法は、vDS ネットワーク環境用です。NSX を使用している場合は、vSphere ポッド を使用してジャンプ ホストを作成します。プライベート キーを使用した、システム ユーザーとしての TKG サービス クラスタ ノードへの SSH 接続を参照してください。

1 vSphere Client を使用して vCenter Server にログインします。

2 新しい仮想マシンを作成します。

3 Linux ゲスト OS を選択します。この例では、VMware Photon OS（64 ビット）を選択します。

4 OS をインストールします。これを行うには、ISO ファイルをダウンロードして仮想マシンに適用し、起動します。

5 [ワークロード ネットワーク] > [名前空間ネットワーク] で、IP アドレスを使用して仮想マシンを構成します。

注： ワークロード ネットワークの範囲が、使用しているポート グループのネットワーク領域全体と重なる場合は、IP アドレスが競合しても構いません。

6 2 つ目の仮想 NIC を仮想マシンに追加し、管理ネットワークまたはフロントエンド ネットワークに割り当てます。

7 OS の構成を完了し、再起動後に仮想マシンをパワーオンします。

8 仮想マシンの vSphere コンソールに root ユーザーとしてログインします。

9 新しい NIC のネットワーク インターフェイスを作成して、フロントエンド ネットワーク上の IP アドレスを割り当てます。

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

注： この方法による効果は、再起動すると無効になります。

10 このインターフェイスからゲートウェイと DNS サーバに ping を実行できることを確認します。

11 仮想マシンの vSphere コンソールで、証明書を使用して SSH ユーザーを設定します。ネストされたシェルを作成して、機能することを確認します。

12 SSH ユーザーとしてフロントエンド ネットワークからジャンプ ボックスに SSH 接続して、正常に動作することを確認します。

13 sshpass を仮想マシンにインストールします（これにより、パスワードを使用して SSH 接続からログインできるようになります）。PhotonOS の場合、コマンドは次のとおりです。

```
tdnf install -y sshpass
```

14 SSH がパスワードなしで動作するように、`~/.ssh/authorized_keys` ファイルにクライアントのパブリック キーを追加し、`sshd` プロセスを再起動します。

- パブリック キーを取得します（例：`cat ~/.ssh/id_rsa.pub`）。
- ジャンプ ホスト仮想マシンにアクセスします。
- SSH ディレクトリがない場合は、次のコマンドで作成します：`mkdir -p ~/.ssh`。
- `authorized_keys` ファイルにパブリック キーを追加します。`echo ssh-rsa AAAA.... >> ~/.ssh/authorized_keysssh-rsa AAAA....` を、`cat ~/.ssh/id_rsa.pub` コマンドで出力されたパブリック キー文字列全体に置き換えます。
- `~/.ssh` ディレクトリと `authorized_keys` ファイルに適切な権限が設定されていることを確認します（例：`chmod -R go= ~/.ssh`）。

プラットフォーム オペレータ専用のグループおよびロールの作成

スーパーバイザー および TKG サービス クラスタの運用に責任を負う担当者などの vSphere IaaS control plane オペレータに権限を割り当てる方法の 1 つは、専用の vSphere ユーザー グループと、そのようなオペレータ専用のロールを作成することです。

プラットフォーム オペレータ グループおよびロールについて

セキュリティ上のベスト プラクティスとして、vSphere 管理者ロールをプラットフォーム オペレータに割り当てないようにすることを推奨します。管理者ロールは、TKG サービス クラスタを操作するために必要な権限よりも多くの権限が付与されるためです。最小限の権限の原則に従って、プラットフォーム オペレータが使用する専用のユーザー グループ、サービス (ユーザー) アカウント、カスタム ロールを作成してから、そのユーザー グループに vSphere オブジェクトに対するカスタム ロール権限を付与することができます。

注： このトピックのすべてのタスクを実行するには、vSphere 管理者として vCenter Server にログインする必要があります。

注： vSphere グループ名およびロール名は、ユーザー定義の文字列です。ここに記載されている名前は例であり、実際のセキュリティおよびビジネスのニーズに応じて採用、調整、または変更できます。

注意: ここで提供されているロール権限の例は、実際のセキュリティ要件とビジネス要件のコンテキストで評価し、ロールが確実にコンプライアンスを満たしているか確認して、必要に応じて調整する必要があります。ここで使用されているすべての権限がニーズに適合するとは限りません。また、追加の権限が必要になる場合もあります。vSphere の権限とセキュリティに関する考慮事項の完全なリストについては、vSphere セキュリティのドキュメントを参照してください。

パート 1：プラットフォーム オペレータ グループとユーザーの作成

vCenter Server、または vCenter Server と統合された AD/LDAP システムで、プラットフォーム オペレータ グループと初期ユーザー アカウントを作成します。

1 vSphere Client を使用して vCenter Server に管理者としてログインします。

2 [管理] - [Single Sign-On] - [ユーザーおよびグループ] の順に移動します。

3 [グループ] タブを選択します。

4 [追加] をクリックし、新しいグループを作成します。

- 名前：*platform-operators-group*
- 説明：*スーパーバイザーおよび TKG サービス クラスタの Kubernetes オペレータ用グループ アカウント*
- [追加] をクリックします。

5 [ユーザー] タブを選択します。

6 [追加] をクリックし、テスト用の新しいユーザーを作成します。

- 名前：*platform-operator-00*
- パスワード：要件を満たす強力なパスワードを入力します
- [[追加]] をクリックします。

7 [グループ] タブを選択します。

8 新しいユーザーをグループに追加します。

- グループ *platform-operators-group* を選択します。

- [メンバーの追加] をクリックします。
- [vsphere.local] を選択します。
- ユーザー名 *platform-operator-00* を検索します。
- このユーザーを選択し、[追加] をクリックします。
- [保存] をクリックします。

パート 2：サービス プロバイダ ユーザー グループへのプラットフォーム オペレータ グループの追加

サービス プロバイダ ユーザー グループにプラットフォーム オペレータ グループを追加します。これにより、プラットフォーム オペレータ グループのメンバーは、vCenter Server の [インベントリ] - [ホストおよびクラスタ] 画面で vSphere 名前空間 を表示できるようになります。

1 [管理] - [Single Sign-On] - [ユーザーおよびグループ] の順に移動します。

2 [グループ] タブを選択します。

3 [ServiceProviderUsers] グループを見つけます。

4 [ServiceProviderUsers] グループを編集し、[platform-operators-group] のメンバーとして追加します。

5 [保存] をクリックします。

パート 3：プラットフォーム オペレータ ロールの作成

プラットフォーム オペレータ用のカスタム vCenter SSO ロールを作成します。

注： このロールには、スーパーバイザー および TKG サービス クラスタをプロビジョニングし、コンテンツ ライブラリの管理を含めて運用するために必要なすべての権限が含まれています。ビジネス要件とセキュリティ要件に基づいて、このロールに割り当てられている権限の調整が必要になる場合があります。また、ロールが要件を満たしていることを確認するために、ロールをテストする必要があります。

1 vSphere Client を使用して、[管理] - [アクセス コントロール] - [ロール] の順に移動します。

2 [新規] を選択し、新しいロール名 **`platform-operators-role`** を作成します。

3 このロールに対して次の権限を定義します。

4 完了したら、[保存] をクリックします。

```
- 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 &
```

パート 4：プラットフォーム オペレータ グループおよびロールへの vCenter Server オブジェクト権限の割り当て

スーパーバイザー および TKG サービス クラスタが使用する vCenter Server オブジェクトに対する権限をプラットフォーム オペレータ グループに割り当てます。

1 vCenter Server で、[インベントリ] ビューを選択します。

2 次のリストにある各 vCenter Server オブジェクトについて、オブジェクトを右クリックし、[権限の追加] を選択します。

3 [ユーザー/グループ] で、グループ [platform-operators-group] を選択します。

4 [ロール] で、ロール [platform-operators-group-role] を選択します。

5 一部のオブジェクトでは、[子に伝播] を選択する必要があります。

- [ホストおよびクラスタ]
  - ルート vCenter Server オブジェクト。
  - データセンター オブジェクトから始まり TKG デプロイを管理するクラスタに至るまでの、データセンターおよびすべてのホストおよびクラスタ フォルダ。
  - [子に伝播] が有効になっている、スーパーバイザー が有効なターゲット vCenter Server クラスタ（ESXi ホストの場合など）。
  - [子に伝播] が有効になっているターゲット リソース プール。
- [仮想マシンおよびテンプレート]
  - [子に伝播] が有効になっている最上位のデータセンター オブジェクト。
  - または、より細かく設定する場合、[子に伝播] が有効になっているターゲット仮想マシンおよびテンプレート フォルダ。
- [ストレージ]
  - [子に伝播] が有効になっている最上位のデータセンター オブジェクト。
  - または、[子に伝播] が有効になっていない共有データストア オブジェクト（vsanDatastore など）、または [子に伝播] が有効になっている、個々のデータストアおよびすべてのストレージ フォルダ（データセンター オブジェクトから始まり TKG のデプロイに使用されるデータストアに至るまで）。
- [ネットワーク]
  - [子に伝播] が有効になっている最上位のデータセンター オブジェクト。

- または、クラスタの割り当て先となる個々のネットワーク、分散スイッチ、分散ポート グループ。

パート 5：プラットフォーム オペレータ グループとロールの関連付け

プラットフォーム オペレータ グループとロールに権限を割り当てます。

1 vSphere Client を使用して、[管理] - [アクセス コントロール] - [グローバル権限] - [権限の追加] の順に移動します。

2 プラットフォーム オペレータ グループにプラットフォーム オペレータ ロールを追加します。
   - [追加] をクリックします。
   - [ドメイン] で、[vsphere.local] を選択します。
   - [ユーザー/グループ] に、グループ [platform-operators-group] を入力します。
   - [ロール] で、ロール [platform-operators-role] を選択します。
   - [子へ伝達] チェック ボックスをオンにします。
   - [OK] をクリックして、グループのロール権限を更新します。

3 プラットフォーム オペレータ グループに vSphere Kubernetes マネージャ ロールを追加します。
   - [追加] をクリックします。
   - [ドメイン] で、[vsphere.local] を選択します。
   - [ユーザー/グループ] に、グループ [platform-operators-group] を入力します。
   - [ロール] で、[vSphere Kubernetes マネージャ] ロールを選択します。
   - [子へ伝達] チェック ボックスをオンにします。
   - [OK] をクリックして、グループのロール権限を更新します。

パート 6：プラットフォーム オペレータ グループとロールの検証

新しいプラットフォーム オペレータ グループとロールをテストします。グループとロールがセキュリティ要件とビジネス要件を満たしていることを確認し、必要に応じて調整する必要があります。

1 vSphere Client を使用して、[platform-operator-00] ユーザー アカウントで vCenter Server にログインします。

2 ホストおよびクラスタ、仮想マシンおよびテンプレート、ストレージ、ネットワークなどの vSphere オブジェクトに対する読み取りアクセス権があることを確認します。

3 コンテンツ ライブラリを作成および構成できることを確認します。5 章 TKG サービス クラスタ用 Kubernetes リリースの管理を参照してください。

4 vSphere 名前空間 を作成可能で、ユーザーの追加、コンテンツ ライブラリの関連付け、ストレージ ポリシーの割り当てなどの構成ができることを確認します。6 章 TKG サービス クラスタをホストするための vSphere 名前空間 の構成を参照してください。

5 vSphere 向け Kubernetes CLI Tools を使用して スーパーバイザー にログインできることを確認します。vCenter SSO 認証を使用した TKG サービス クラスタへの接続を参照してください。

6 Kubectl を使用して スーパーバイザー に TKG クラスタをプロビジョニングできることを確認します。Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。

7 Kubectl を使用して スーパーバイザー 上の TKG クラスタにログインできることを確認します。Kubectl を使用した vCenter Single Sign-On ユーザーとしての TKG サービス クラスタへの接続を参照してください。

8 スーパーバイザー 上の TKG クラスタにワークロードをデプロイできることを確認します。12 章 TKG サービス クラスタへのワークロードのデプロイを参照してください。

9 スーパーバイザー 上の TKG クラスタに対してさまざまな操作タスクを実行できることを確認します。8 章 TKG サービス クラスタの操作を参照してください。

10 vSphere Client を使用して、vSphere 名前空間 内で スーパーバイザー にプロビジョニングされた TKG クラスタを表示できることを確認します。

11 vSphere Client を使用して、スーパーバイザー および TKG クラスタ オブジェクトを監視できることを確認します。

12 ネガティブ テストを実行して、vSphere 管理者用に予約された特定のタスクを実行できないことを確認します。たとえば、新しい vSphere ストレージ ポリシーや vSphere ネットワークの作成はできないようにします。また、ワークロード管理を無効にすることもできないようにします。

13 セキュリティ要件およびビジネス要件を満たすように、ロールの権限を適切に調整します。

TKG サービス クラスタ用 Kubernetes リリースの管理

5

Tanzu Kubernetes リリース (TKr) は、TKG サービス クラスタ用に Kubernetes ソフトウェア ディストリビューションを提供します。TKr は、プラットフォームと同期する仮想マシン テンプレートとして、vCenter Server コンテンツ ライブラリを使用して VMware によって配布されます。

次のトピックを参照してください。

- TKG サービス クラスタでの Kubernetes リリースの使用
- コンテンツ ライブラリを管理するために必要なロール権限
- 新しいサブスクライブ済みコンテンツ ライブラリの作成
- ローカル コンテンツ ライブラリの作成（エアギャップ クラスタのプロビジョニング用）
- ローカル コンテンツ ライブラリの公開の有効化
- 既存のコンテンツ ライブラリの編集
- コンテンツ ライブラリの移行
- TKr の解決の概要

TKG サービス クラスタでの Kubernetes リリースの使用

Tanzu Kubernetes リリース (Tkr) は、TKG サービス クラスタで使用するための、VMware によって署名およびサポートされた Kubernetes ソフトウェア ディストリビューションを提供します。TKr 形式は、vSphere 8 でパッケージと複数のオペレーティング システムがサポートされるように更新されています。

TKr リリース ノート

使用可能な TKr の詳細なリスト、各リリースの新機能、既知の問題、および TKr の互換性については、「Tanzu Kubernetes リリースのリリース ノート」を参照してください。

TKr の配布と使用

VMware は、コンテンツ配信ネットワークを介して Tanzu Kubernetes リリース を配布します。vSphere コンテンツ ライブラリを使用して、TKr と vSphere 名前空間 の関連付けを行います。TKr を自動的に使用するには、新しいサブスクライブ済みコンテンツ ライブラリの作成を使用します。インターネットが制限されている環境では、ローカル コンテンツ ライブラリの作成（エアギャップ クラスタのプロビジョニング用）を使用します。

各 Tanzu Kubernetes リリース は OVA テンプレートとして配布されます。スーパーバイザー の TKr コントローラは、OVA テンプレートを使用して TKG クラスタ ノードの仮想マシンを構築します。仮想マシンのディスク サイズは TKr OVA テンプレートによって設定されます。CPU および RAM リソースは、TKG サービス クラスタでの仮想マシン クラスの使用を使用して指定します。

TKr コンテンツ ライブラリは、名前空間の範囲内にありません。すべての vSphere 名前空間 は、スーパーバイザー を使用する TKG に対して同じ TKr コンテンツ ライブラリを使用します。1 つの vSphere 名前空間 の TKr コンテンツ ライブラリを編集すると、他のすべての名前空間の TKr コンテンツ ライブラリも更新されます。

TKr NAME 文字列

コマンド `kubectl get tkr` を使用して、vSphere 名前空間 で使用可能な TKr イメージを一覧表示します。例：

```
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
```

TKr NAME 文字列を使用して、スーパーバイザー 上に TKG クラスタをプロビジョニングします。TanzuKubernetesCluster v1alpha3 API を使用している場合は、`tkr.reference.name` フィールドに完全な TKr NAME 文字列を指定します。クラスタ v1beta1 API を使用している場合は、`topology.version` フィールドに完全な TKr NAME 文字列を指定します。

注： クラスタ仕様で TKr を参照する場合は、version 文字列を使用しないでください。形式は TKr NAME 文字列と正確に一致する必要があります。

コンテンツ ライブラリでの TKr の名前は、完全な TKr NAME 文字列である必要があります。サブスクライブ済みコンテンツ ライブラリを使用している場合は、TKr NAME 文字列が作成されます。ローカル コンテンツ ライブラリを使用している場合は、TKr に付与する名前が TKr NAME 文字列と一致していることを確認します。詳細については、ローカル コンテンツ ライブラリの作成（エアギャップ クラスタのプロビジョニング用）を参照してください。

TKr の TKG サービス との互換性

TKr は、TKG サービス および スーパーバイザー とは別個にリリースおよび更新されます。

TKG クラスタをプロビジョニングするには、TKr に TKG サービス との互換性が必要です。TKG サービス との互換性がない TKr は使用できません。また、アップグレード対象のバージョンと互換性のある TKr を実行していることを確認する必要があります。

TKr の互換性を確認するには、コマンド `kubectl get tkr` を使用します。COMPATIBLE 列はブール値を返します。`True` は TKr に互換性があることを意味し、`False` は TKr に互換性がないことを意味します。

TKr の vSphere との互換性

vSphere 8 では TKr 形式が更新されています。vSphere 8 の TKr は、vSphere 8.x でのみ実行できます。vSphere 7.x の TKr は、vSphere 7 で動作するレガシー イメージです。このようなイメージは vSphere 8 で実行できますが、アップグレードを目的とする場合に限られます。レガシー TKr イメージは、`legacy-tkr` 注釈ラベルによって識別されます。

TKr の互換性を確認するには、コマンド `kubectl get tkr -o yaml` と `kubectl get tkr --show-labels` を使用します。注釈ラベル `legacy-tkr` が存在する場合、TKr は vSphere 8 の機能をサポートしないため、vSphere 7 から vSphere 8 へのアップグレードに限り使用する必要があります。

たとえば、次のコマンドは、指定したイメージが `legacy-tkr` であることを示しています。

```
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
```

次の例では、`--show-labels` フラグを使用して TKr の互換性を確認します。ラベル `legacy-tkr` があるため、イメージはレガシー TKG クラスタの作成にのみ使用できます。

```
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=],
```

次の例では、ラベル `legacy-tkr` がラベル リストに含まれていないため、TKr が vSphere 8.x 専用であることが示されています。

```
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=
```

また、`-o yaml` を使用して同等なコマンドを実行した場合も、`legacy-tkr` ラベルは存在せず、TKr が vSphere 8.x 専用であることが示されます。

```
 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 オペレーティング システムのイメージ形式

TKr OS イメージ形式では、1 つの TKr に対して複数のオペレーティング システム イメージがサポートされています。つまり、サポートされているすべてのオペレーティング システム（現在は PhotonOS と Ubuntu）の特定の Kubernetes バージョンに対して、Tanzu Kubernetes リリース が 1 つあります。デフォルトの OS イメージ形式は PhotonOS です。

デフォルトでは、TKG クラスタ ノードには、指定した TKr の PhotonOS エディションが使用されます。参照されている TKr で OS イメージ形式がサポートされていて、Ubuntu OS エディションが使用可能な場合は、`run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu` 注釈を使用して TKr の Ubuntu OS エディションを指定します。たとえば、次の 7 章 TKG サービス クラスタのプロビジョニングでは、TKr という名前の Ubuntu エディションを使用しています。

```
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
```

OS イメージ形式は、異種クラスタのデプロイをサポートしています。たとえば、次のクラスタ マニフェストは、制御プレーン ノード用にデフォルトの PhotonOS、ワーカー ノード用に Ubuntu を使用する Tanzu Kubernetes クラスタを作成します。TKr のバージョンは制御プレーンセクションで参照されています。名前付きワーカー ノード プールを使用する場合は、注釈で Ubuntu が指定されます。

```
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
```

システムを vSphere 8 にアップグレードすると、既存の TKr は、単一の OS イメージを参照して TKr OS イメージ形式に自動的に変換されます。これにより、レガシー TKr を非レガシー TKr にアップグレードする場合の互換性を確保できます。

レガシー TKr には、Photon と Ubuntu の 2 つのエディションがあります。レガシー TKr に Ubuntu 固有のエディションがある場合は、バージョン文字列全体（「ubuntu」を含む）を使用して注釈ラベルを省略するか、バージョン文字列の短縮形（「ubuntu」を含まない）を使用してバージョン ラベルを含めることができます。

TKr パッケージ

vSphere 8 と互換性がある TKr イメージは、コンテナ ストレージ インターフェイス (CSI) やコンテナ ネットワーク インターフェイス (CNI) などのコア コンポーネント用のパッケージベースのフレームワークに更新されます。v1beta API を使用している場合、これらのコンポーネントの変更または更新はカスタム リソース定義を使用して行われます。

TKr を構成するパッケージを表示するには、次のコマンドを実行します。

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

例：

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

このコマンドは、TKr 内のすべてのパッケージを返します。例：

```
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
```

使用事例の例については、v1beta1 の例：Calico CNI を含むクラスタを参照してください。

TKr OS タイプの移行

TKr オペレーティング システム間でクラスタのインプレース アップデートを行うことはできません。つまり、TKr バージョン 1.27 Photon を使用している TKG クラスタを TKr v1.28.8 Ubuntu にアップグレードすることなどはできません。

TKG クラスタが使用している TKr OS タイプを変更する場合は、次の手順を検討してください。この例では、元のクラスタは TKr Photon を使用していて、アップグレード先は TKr Ubuntu です。

- Velero を使用して、Photon ベースの TKG クラスタ ワークロードをバックアップします。

  20 章 TKG サービス クラスタとワークロードのバックアップとリストアを参照してください。

- Ubuntu TKr を使用して新しい TKG クラスタをプロビジョニングします。

  7 章 TKG サービス クラスタのプロビジョニングを参照してください。

- Velero を使用して、TKG クラスタ ワークロードを Ubuntu クラスタにリストアします。

  20 章 TKG サービス クラスタとワークロードのバックアップとリストアを参照してください。

TKr のセキュリティ強化

スーパーバイザー と TKr を含むシステム コンポーネントのセキュリティ技術導入ガイド (STIG) を利用できます。詳細については、「Tanzu STIG Hardening」を参照してください。

独自の TKr の構築

TKr v1.25.7 for vSphere 8.x 以降では、vSphere 8 で TKG クラスタのカスタム TKr マシン イメージを構築できます。カスタム マシン イメージには、サポートされているオペレーティング システムとバージョン、リリースされた TKr に基づく Kubernetes バージョン、およびユーザーが行ったカスタマイズがパッケージ化されています。

TKG クラスタ ノード用のカスタム マシン イメージを構築するには、vSphere Tanzu Kubernetes Grid Image Builder を使用します。カスタム イメージの構築、サポートされている TKr バージョン、およびサポートされているカスタマイズの詳細については、ドキュメントを参照してください。

コンテンツ ライブラリを管理するために必要なロール権限

コンテンツ ライブラリを管理するには、vSphere ロール権限がいくつか必要です。vSphere 管理者ロールを使用していない場合は、必要に応じて権限のリストを参照してください。

コンテンツ ライブラリの管理に必要な権限

コンテンツ ライブラリの管理には、次の vSphere ロール権限が必要です。

これらの権限は、vSphere 管理者ロールに含まれています。

これらの権限を含む専用ロールを作成するには、「プラットフォーム オペレータ専用のグループおよびロールの作成」を参照してください。

```
- 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 &
```

新しいサブスクライブ済みコンテンツ ライブラリの作成

TKG クラスタを スーパーバイザー にプロビジョニングするには、サブスクライブ済みコンテンツ ライブラリを作成し、Tanzu Kubernetes リリース を同期します。サブスクライブ済みコンテンツ ライブラリを使用すると、TKR のディストリビューションを自動化し、最新のリリースで最新の状態を維持できます。

スーパーバイザー 上の TKG の場合、VMware は Tanzu Kubernetes リリース をコンテンツ配信ネットワークに公開します。これらのイメージ公開をサブスクライブするコンテンツ ライブラリを作成できます。同期モード（即時またはオンデマンド）を選択します。

注意: パブリック コンテンツ配信ネットワーク上の Tanzu Kubernetes リリース の即時同期には、かなりの時間とディスク容量が必要になる可能性があります。

前提条件

コンテンツ ライブラリ機能は、スーパーバイザー 上の TKG が依存する vCenter Server の機能です。詳細については、vSphere の仮想マシン管理の「コンテンツ ライブラリの使用」を参照してください。

手順

1 vSphere Client を使用して、vCenter Server にログインします。

2 [メニュー] - [コンテンツ ライブラリ] の順に選択します。

3 [作成] をクリックします。

[新しいコンテンツ ライブラリ] ウィザードが開きます。

4 コンテンツ ライブラリの [名前と場所] を指定し、完了したら [次へ] をクリックします。

フィールド	説明
名前	`TKR-sub` などのわかりやすい名前を入力します。
メモ	`TKG2 の TKR 用サブスクリプション ライブラリ`などの説明を入力します。
vCenter Server	[ワークロード管理] が有効になっている vCenter Server インスタンスを選択します。

5 [コンテンツ ライブラリの設定] 画面でコンテンツ ライブラリのサブスクリプションを構成し、完了したら [次へ] をクリックします。

a [サブスクライブ済みコンテンツ ライブラリ] オプションを選択します。

b 発行者の[サブスクリプション URL] アドレスを入力します。

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

c [コンテンツのダウンロード] オプションで、次のいずれかを選択します。

オプション	説明
ただちに	サブスクリプション プロセスによって、ライブラリ メタデータとイメージの両方が同期されます。公開ライブラリからアイテムが削除されても、そのコンテンツはサブスクライブ済みライブラリのストレージに残るため、手動で削除する必要があります。
必要に応じて	サブスクリプション プロセスでは、ライブラリ メタデータのみが同期されます。イメージが公開されると、Tanzu Kubernetes Grid Services によってダウンロードされます。アイテムが不要になったら、アイテムのコンテンツを削除してストレージの容量を解放できます。ストレージを保存するには、このオプションを推奨します。

6 プロンプトが表示されたら、SSL 証明書のサムプリントを受け入れます。

SSL 証明書のサムプリントは、インベントリから購読済みコンテンツ ライブラリを削除するまで、システムに格納されます。

7 [セキュリティ ポリシーの適用] 画面で OVF セキュリティ ポリシーを構成し、完了したら [次へ] をクリックします。

a [セキュリティ ポリシーの適用] を選択します。

b [OVF デフォルト ポリシー] を選択します。

このオプションを選択すると、同期プロセス中に OVF 署名証明書が検証されます。証明書の検証に合格しなかった OVF テンプレートは、[検証失敗] タグでマークされます。テンプレートのメタデータは保持されますが、OVF ファイルは同期できません。

注： 現在サポートされているセキュリティ ポリシーは、[OVF デフォルト ポリシー] のみです。

8 [ストレージの追加] 画面で、コンテンツ ライブラリのコンテンツのストレージ場所としてデータストアを選択し、[次へ] をクリックします。

9 [設定の確認] 画面で詳細を確認し、[完了] をクリックします。

10 [コンテンツ ライブラリ] 画面で、作成した新しいコンテンツ ライブラリを選択します。

11 ライブラリ コンテンツの同期を確認または完了します。

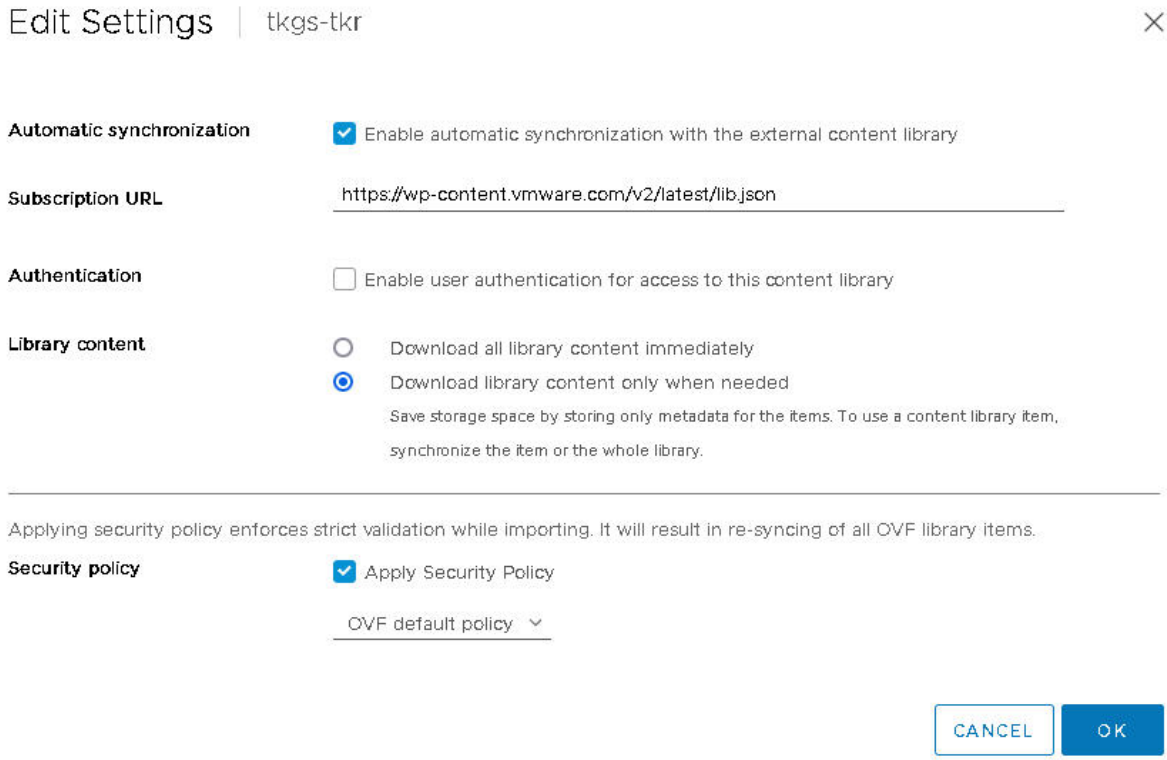
同期オプション	説明
ただちに	すべてのコンテンツをすぐにダウンロードする場合は、ライブラリが同期されていることを確認します。 同期されたライブラリ コンテンツを表示するには、[テンプレート] - [OVF & OVA テンプレート] の順に選択します。
必要に応じて	必要に応じてライブラリを同期する場合は、次の 2 つの方法があります。 ■ [アクション] > [同期] を使用してライブラリ全体を同期する ■ アイテムを右クリックし、[同期] を選択して、このアイテムのみを同期します。 同期されたライブラリ コンテンツを表示するには、[テンプレート] - [OVF & OVA テンプレート] の順に選択します。

12 [必要に応じて] オプションを選択した場合は、使用する OVF テンプレートをダウンロードします。

[必要に応じて] オプションを選択した場合、イメージ ファイルはローカルに保存されず、メタデータのみが保存されます。テンプレート ファイルをダウンロードするには、アイテムを選択して右クリックし、[アイテムの同期] を選択します。

13 サブスクライブ済みコンテンツ ライブラリの設定を更新するには、[アクション] - [設定の編集] の順に選択します。

設定	値
サブスクリプション URL	`https://wp-content.vmware.com/v2/latest/lib.json`
認証	有効になっていません
ライブラリ コンテンツ	必要に応じてダウンロード
セキュリティ ポリシー	OVF デフォルト ポリシー



次のステップ

Tanzu Kubernetes リリース コンテンツ ライブラリは、TKG クラスタをプロビジョニングする各 vSphere 名前空間 と関連付ける必要があります。6 章 TKG サービス クラスタをホストするための vSphere 名前空間 の構成を参照してください。

ローカル コンテンツ ライブラリの作成（エアギャップ クラスタのプロビジョニング用）

TKG クラスタを スーパーバイザー にプロビジョニングするには、ローカル コンテンツ ライブラリを作成し、Tanzu Kubernetes リリース をインポートします。ローカル コンテンツ ライブラリの一般的な使用事例は、インターネットが制限されている（エアギャップ）環境です。

ローカル コンテンツ ライブラリを作成するには、ライブラリの構成、OVA ファイルのダウンロード、OVA ファイルのローカル コンテンツ ライブラリへのインポートを行います。

前提条件

コンテンツ ライブラリ機能は、スーパーバイザー 上の TKG が依存する vCenter Server の機能です。詳細については、コンテンツ ライブラリの使用を参照してください。

手順

1 vSphere Client を使用して、vCenter Server にログインします。

2 [メニュー] をクリックします。

3 [コンテンツ ライブラリ] をクリックします。

4 [作成] をクリックします。

[新しいコンテンツ ライブラリ] ウィザードが表示されます。

5 コンテンツ ライブラリの [名前と場所] を指定し、完了したら [次へ] をクリックします。

フィールド	説明
名前	`TKr-local` などのわかりやすい名前を入力します。
メモ	`TKG の TKr 用ローカル ライブラリ`などの説明を入力します。
vCenter Server	vSphere IaaS control plane が有効になっている vCenter Server インスタンスを選択します。

6 [コンテンツ ライブラリの設定] 画面で、[ローカル コンテンツ ライブラリ] オプションを選択し、[次へ] をクリックします。

ローカル コンテンツ ライブラリの場合、使用する OVF テンプレートを手動でインポートします。

7 [セキュリティ ポリシーの適用] 画面で OVF セキュリティ ポリシーを構成し、完了したら [次へ] をクリックします。

a [セキュリティ ポリシーの適用] を選択します。

b [OVF デフォルト ポリシー] を選択します。

このオプションを選択すると、同期プロセス中に OVF 署名証明書が検証されます。証明書の検証に合格しなかった OVF テンプレートは、[検証失敗] タグでマークされます。テンプレートのメタデータは保持されますが、OVF ファイルは同期できません。

注： 現在サポートされているセキュリティ ポリシーは、[OVF デフォルト ポリシー] のみです。

8 [ストレージの追加] 画面で、コンテンツ ライブラリのコンテンツのストレージ場所としてデータストアを選択し、[次へ] をクリックします。

9 [設定の確認] 画面で詳細を確認し、[完了] をクリックします。

10 [コンテンツ ライブラリ] 画面で、作成した新しいコンテンツ ライブラリを選択します。

11 ローカル コンテンツ ライブラリにインポートする各 Tanzu Kubernetes リリース の OVA ファイルをダウンロードします。

a ブラウザを使用して、次の URL に移動します。

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

b 目的のイメージのディレクトリをクリックします。通常、このディレクトリは、Kubernetes ディストリビューションの最新または直近のバージョンです。

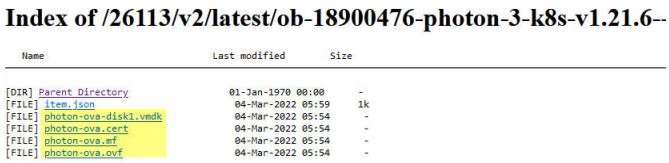
例：

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

重要： ローカル コンテンツ ライブラリにファイルをインポートするには、ディストリビューション名を使用する必要があります。ターゲット名をファイルにコピーするか、この手順を完了するまでブラウザを開いたままにしてください。上記の例に基づいて、必要となる名前の文字列の必須部分は `photon-3-k8s-v1.20.7---vmware.1-tkg.1.7fb9067` です。

c 次の各ファイルに対して、右クリックして [リンクに名前を付けて保存] を選択します。

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



d 各ファイルがローカル ファイル システムに正常にダウンロードされていることを確認します。

注： インポートするファイルは、OVF ファイルと VMDK ファイルです。ただし、セキュリティ ポリシーが適用されている場合は、証明書 (`*.cert`) とマニフェスト (`*.mf`) を含む 4 つのすべてのファイルがインポート時にソース ディレクトリに存在する必要があります。証明書ファイルとマニフェスト ファイルがインポート時にない場合、インポートされた Tanzu Kubernetes リリースは使用できません。

12 OVA ファイルと VMDK ファイルをローカル コンテンツ ライブラリにインポートします。

a [メニュー] - [コンテンツ ライブラリ] - [ ] の順に選択します。

b [コンテンツ ライブラリ] のリストから、作成したローカル コンテンツ ライブラリの名前のリンクをクリックします。

c [アクション] をクリックします。

d [アイテムのインポート] を選択します。

e [ライブラリ アイテムのインポート] ウィンドウで、[ローカル ファイル] を選択します。

f [ファイルのアップロード] をクリックします。

g `photon-ova.ovf` ファイルと `photon-ova-disk1.vmdk` ファイルの両方を選択します。

`2 files ready to import` というメッセージが表示されます。各ファイルには、名前の横に緑色のチェック マークが表示されます。

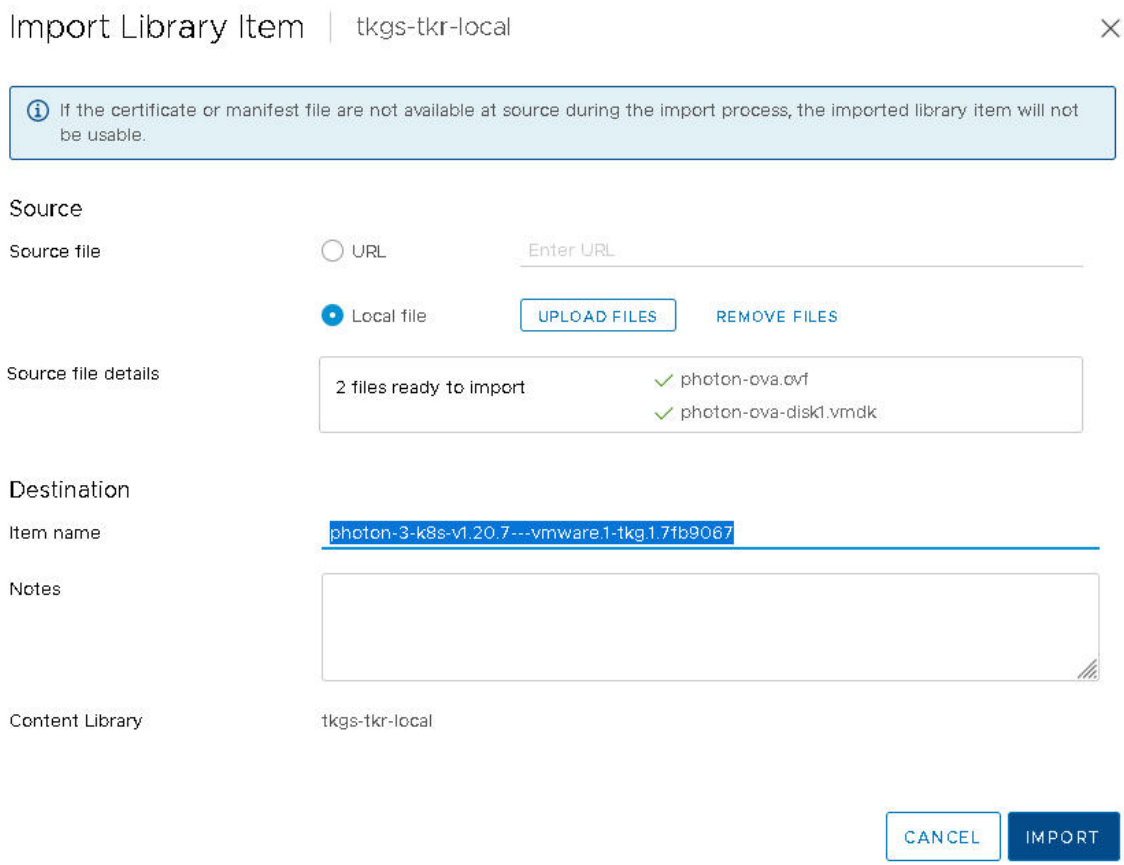
h ファイルをダウンロードしたディレクトリにある OS イメージのバージョンと Kubernetes バージョンを合わせたものになるように、[ターゲット アイテム] 名を変更します。

例：

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

注意: コンテンツ ライブラリの [ターゲット アイテム] 名は、目的の Tanzu Kubernetes リリースのフォルダ名の文字列と正確に一致する必要があります。名前が一致しない場合、スーパーバイザー はイメージを有効な Tanzu Kubernetes リリースに解決できません。

i [インポート] をクリックします。



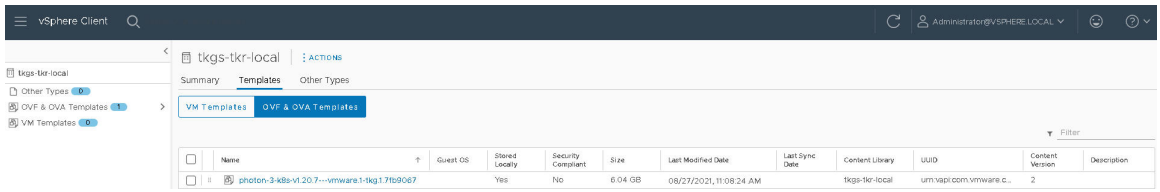
13 ローカル コンテンツ ライブラリに Tanzu Kubernetes リリース がポピュレートされることを確認します。

a 画面の下部にある [最近のタスク] ペインを表示します。

b [ライブラリ アイテムのコンテンツの取得] タスクを監視し、正常に [完了] になっていることを確認します。

c ローカル コンテンツ ライブラリで、[テンプレート] - [OVF & OVA テンプレート] の順に選択します。

d Tanzu Kubernetes リリース メタデータが一覧表示されて、そのコンテンツがローカルに保存されていることを確認します。



次のステップ

Tanzu Kubernetes リリース コンテンツ ライブラリは、TKG クラスタをプロビジョニングする各 vSphere 名前空間 と関連付ける必要があります。6 章 TKG サービス クラスタをホストするための vSphere 名前空間 の構成を参照してください。

ローカル コンテンツ ライブラリの公開の有効化

ローカル コンテンツ ライブラリを作成し、公開を有効にすると、他のコンテンツ ライブラリからサブスクライブできます。この方法により、TKG クラスタ オペレータにサブスクリプション形式の TKR サービスを提供しながら、使用できる Tanzu Kubernetes リリース イメージを制御することができます。

すべてのイメージが公開され保持されている VMware コンテンツ配信ネットワークを Tanzu Kubernetes リリース のためにサブスクライブする代わりに、Tanzu Kubernetes リリース の限定的なサブセットを提供するローカル コンテンツ ライブラリを作成できます。この方法は、継続的なビルド環境など、使用可能なイメージを制御する場合に便利です。

手順

1 ローカル コンテンツ ライブラリの作成（エアギャップ クラスタのプロビジョニング用）を作成するか、既存のローカル コンテンツ ライブラリの設定を編集します。

2 [公開の有効化] チェックボックスをオンにします。

このローカル コンテンツ ライブラリのサブスクリプション URL を受け取ります。

3 サブスクリプション URL を使用して、1 つ以上の新しいサブスクライブ済みコンテンツ ライブラリの作成を作成します。

結果

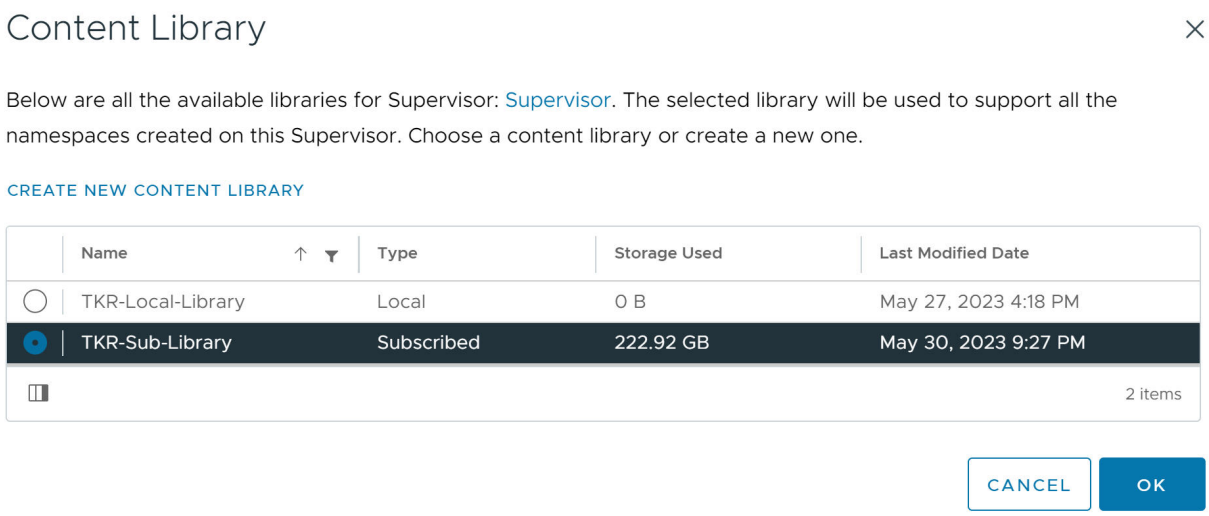
公開が有効になっているローカル コンテンツ ライブラリに Tanzu Kubernetes リリース をアップロードすると、サブスクライブ済みのすべてのコンテンツ ライブラリが同期によってイメージを受信するようになります。イメージのアップロード方法についてはローカル コンテンツ ライブラリの作成（エアギャップ クラスタのプロビジョニング用）のドキュメントを、サブスクライブ済みコンテンツ ライブラリの作成については新しいサブスクライブ済みコンテンツ ライブラリの作成のドキュメントを参照してください。

既存のコンテンツ ライブラリの編集

TKG クラスタに使用する TKR コンテンツ ライブラリの対象範囲は、個々の vSphere 名前空間 ではありません。スーパーバイザー インスタンスのすべての vSphere 名前空間 に同じ TKR コンテンツ ライブラリが使用されます。このため、TKR コンテンツ ライブラリの変更は、スーパーバイザー レベルでのみ許可されます。

vSphere 名前空間 で使用する既存の TKR コンテンツ ライブラリを変更するには、次の手順を実行します。

図 5-1. TKG 用 TKR コンテンツ ライブラリの編集



手順

1 vSphere Client を使用して、vCenter Server にログインします。

2 [ワークロード管理] を選択します。

3 スーパーバイザー インスタンスを選択します。

4 [スーパーバイザー] - [全般] の順に選択します。

5 Tanzu Kubernetes Grid Service を選択します。

6 [コンテンツ ライブラリ] - [編集] の順に選択します。

7 使用する [コンテンツ ライブラリ] を選択し、[OK] をクリックします。

8 新しいコンテンツ ライブラリを作成するには、[新しいコンテンツ ライブラリの作成] を選択します。新しいサブスクライブ済みコンテンツ ライブラリの作成とローカル コンテンツ ライブラリの作成（エアギャップ クラスタのプロビジョニング用）を参照してください。

結果

選択したコンテンツ ライブラリは、スーパーバイザー 上に作成されるすべての vSphere 名前空間 の Tanzu Kubernetes Grid サービスに使用されます。

コンテンツ ライブラリの移行

TKr コンテンツ ライブラリが容量に達した場合は、より多くのストレージ容量を持つ新しいコンテンツ ライブラリに移行できます。

vSphere 管理者がコンテンツ ライブラリを作成する場合、管理者はライブラリ コンテンツを格納するためのデータストア（この場合は OVA ファイル）を指定します。時間の経過とともに配布される Kubernetes バージョンが増え、更新のたびに OVA ファイルが追加されるため、コンテンツ ライブラリのサイズは拡大します。コンテンツ ライブラリには容量が明示的に設定されていませんが、データストアの容量によって制限されています。

コンテンツ ライブラリが容量に達すると、`Internal error occurred: get library items failed for.` というメッセージが表示されることがあります。この場合は、TKG クラスタを新しいコンテンツ ライブラリに移行してストレージ容量を増やすことができます。移行は、vSphere Client を使用して行われます。

手順

1 ターゲット クラスタ用に十分な容量がある新しいコンテンツ ライブラリを作成します。

2 vSphere Client を使用して、vCenter Server にログインします。

3 [メニュー] - [ホストおよびクラスタ] の順に選択します。

4 Tanzu Kubernetes クラスタを含む スーパーバイザー がプロビジョニングされている vSphere クラスタ オブジェクトを選択します。

5 [構成] タブを選択します。

6 ナビゲーション パネルの [名前空間] - [全般] - [] オプションを選択します。

7 メイン パネルの [コンテンツ ライブラリ] セクションの横の [編集] をクリックします。

8 作成した新しいコンテンツ ライブラリを選択し、[OK] をクリックします。

この操作により、クラスタ構成の更新がトリガされます。

注： コンテンツ ライブラリを変更した後、TKG クラスタがコンテンツ ソースから変更を取得できるまでに最大 10 分かかることがあります。

TKr の解決の概要

このトピックでは、システムで TKr イメージを解決する方法について説明します。

TKr の解決

クラスタ オブジェクトが作成または更新される場合、Kubernetes API サーバは Webhook を変更する TKr リゾルバを呼び出します。クラスタ（またはその ClusterClass）には `run.tanzu.vmware.com/resolve-tkr` 注釈が必要です。これがない場合、TKr の解決は完全にスキップされます。TKr リゾルバでは `run.tanzu.vmware.com/resolve-tkr` 注釈の値をラベル クエリとして使用して、候補の TKr のセットを制約します。空の文字列を指定すると、すべての TKr が選択されます。

クラスタ トポロジ controlPlain および machineDeployments の `run.tanzu.vmware.com/resolve-os-image` 注釈の値は、controlPlain と machineDeployments でそれぞれ使用する OSImage オブジェクトのラベル セレクタとして使用されます。解決された TKr によって提供される 1 つの OSImage が、controlPlane またはいずれかの machineDeployments のクエリを満たす必要があります。

指定されたクラスタの spec.topology.version はバージョンのプリフィックスとして使用されます。TKr リゾルバ Webhook では、上記の制約を満たす最新の使用可能な TKR を検索します。見つからない場合は、クラスタの作成/更新要求が拒否されます。

TKR リゾルバ Webhook では、クラスタが次のように変更されます。

- run.tanzu.vmware.com/tkr ラベルが、解決された TKR の名前に設定されます。
- クラスタの spec.topology.version が、解決された TKR の Kubernetes バージョンに設定されます。
- Kubernetes バージョンから controlPlane の TKR および OSImage の値のセットへのマッピングを含めるようにクラスタ変数 TKR_DATA が更新されます。
- Kubernetes バージョンから machineDeployment の TKR および OSImage の値のセットへのマッピングを含めるように、個々の machineDeployments の TKR_DATA 変数のオーバーライドが更新されます。

TKG サービス クラスタをホストするための vSphere 名前空間 の構成

6

vSphere 名前空間 は、TKG サービス クラスタのランタイム環境を提供します。TKG サービス クラスタをプロビジョニングするには、まず、ユーザー、ロール、コンピューティング、ストレージ、コンテンツ ライブラリ、仮想マシンの各クラスを使用して vSphere 名前空間 を構成します。この構成は、その名前空間で実行される TKG サービス クラスタに継承されます。

次のトピックを参照してください。

TKG サービス クラスタでの vSphere 名前空間 の使用

vSphere 名前空間 は、スーパーバイザー のネットワーク範囲内のテナントです。vSphere 名前空間 は TKG サービス クラスタをホストするために使用され、ネットワーク、ロール権限、パーシステント ストレージ、リソース割り当て、およびコンテンツ ライブラリと仮想マシン クラスの統合を提供します。

vSphere 名前空間 ネットワーク

vSphere 名前空間 ネットワークとは、[スーパーバイザー] - [ワークロード ネットワーク] - [名前空間ネットワーク]によって取得されたサブネットです。[名前空間サブネット プリフィックス]は、各 vSphere 名前空間 用に予約されるサブネットのサイズを定義します。デフォルトは /28 です。

vSphere 名前空間 ネットワークは、TKG クラスタと スーパーバイザー の接続を提供します。デフォルトでは、vSphere 名前空間 はクラスタ レベルのネットワーク構成を使用し、そのサブネット内の IP アドレスを割り当てます。vSphere 名前空間 を作成すると、/28 オーバーレイ セグメントおよび対応する IP アドレス プールが、その vSphere 名前空間 内のサービス ポッドにインスタンス化されます。

最初の TKG クラスタが vSphere 名前空間 にプロビジョニングされると、TKG クラスタは vSphere 名前空間 と同じサブネットを共有します。後続の TKG クラスタがその vSphere 名前空間 にプロビジョニングされるたびに、そのクラスタ用の新しいサブネットが作成され、その vSphere 名前空間 ゲートウェイに接続されます。

vSphere 名前空間 には、kubectl トラフィックを各 TKG クラスタ制御プレーンにルーティングする共有ロード バランサ インスタンスがあります。また、TKG クラスタでリソースが提供される Kubernetes サービス ロード バランサごとに、そのサービスのレイヤー 4 ロード バランサ インスタンスが作成されます。

同じ vSphere 名前空間 内の TKG クラスタは、North-South 接続に SNAT IP アドレスを共有します。名前空間の間の East-West 接続は、SNAT ではありません。

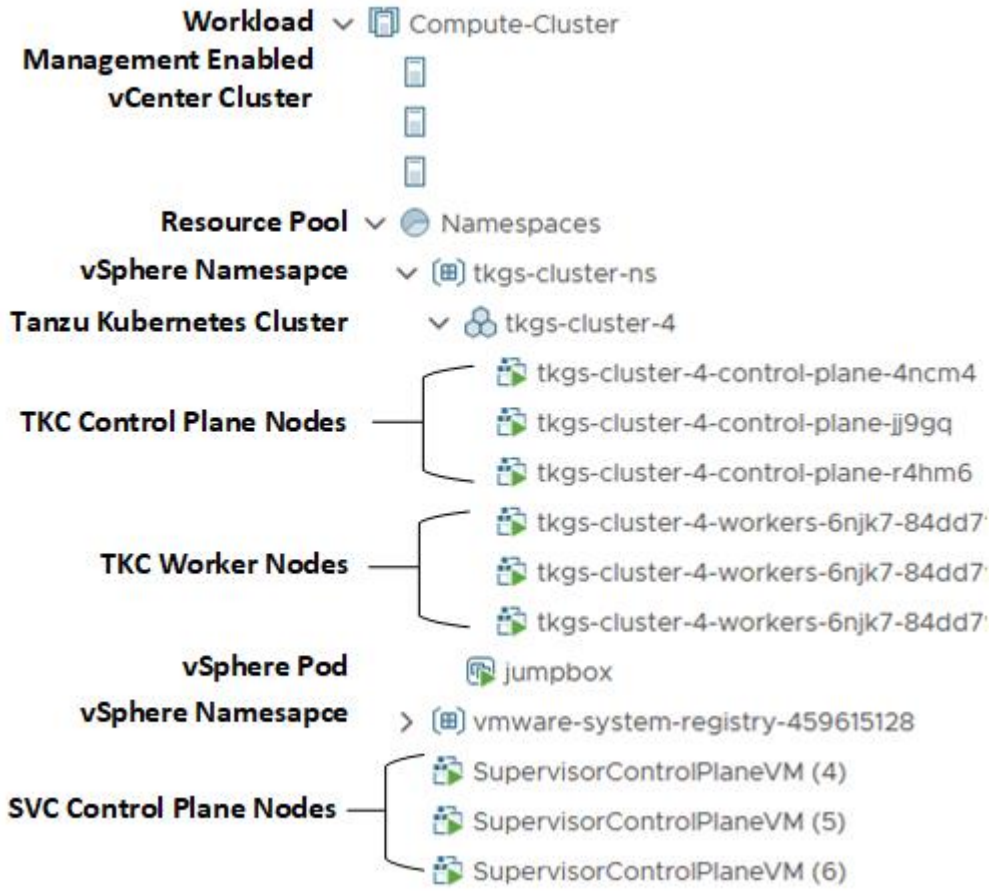
通常、vSphere 名前空間 はルーティングできません。ただし、NSX ネットワークを使用している場合は、ルーティング可能なサブネットで vSphere 名前空間 ネットワークをオーバーライドできます。vSphere 名前空間のワークロード ネットワーク設定のオーバーライドを参照してください。

vSphere 名前空間 リソース プール

単一の vSphere Zone による スーパーバイザー デプロイでは、vSphere 名前空間 を作成すると、その名前空間をバッキングするリソース プールが作成されます。vSphere 名前空間 により、スーパーバイザー 上のコンピューティング、ストレージ、権限、クラス、イメージなどのリソースに論理的な単位が与えられます。たとえば、vSphere 名前空間 に CPU またはメモリの制限を構成すると、その名前空間をバッキングするリソース プールに同じリソース制限が適用されます。このようにして、vSphere 名前空間 により スーパーバイザー でのマルチテナントが可能になります。

3 つの vSphere Zone にデプロイされた スーパーバイザー にも、同じマルチテナント エクスペリエンスが適用されます。ゾーンが設定された スーパーバイザー 上に vSphere 名前空間 を作成すると、その スーパーバイザー をサポートする vSphere クラスタのそれぞれにリソース プールが作成されます。これにより、その vSphere 名前空間 でプロビジョニングされた TKG クラスタを、この スーパーバイザー に属する任意のゾーンにデプロイできます。

vSphere Client を使用しているときに、vSphere 名前空間 リソース プールおよびオブジェクトを表示するには、[ホストおよびクラスタ] パースペクティブを選択し、[仮想マシンおよびテンプレート] ビューも選択します。TKG クラスタをプロビジョニングすると、クラスタはターゲット vSphere 名前空間 に作成されます。ゾーンが設定された スーパーバイザー デプロイでは、各 vSphere クラスタで同じリソース プールを使用します。



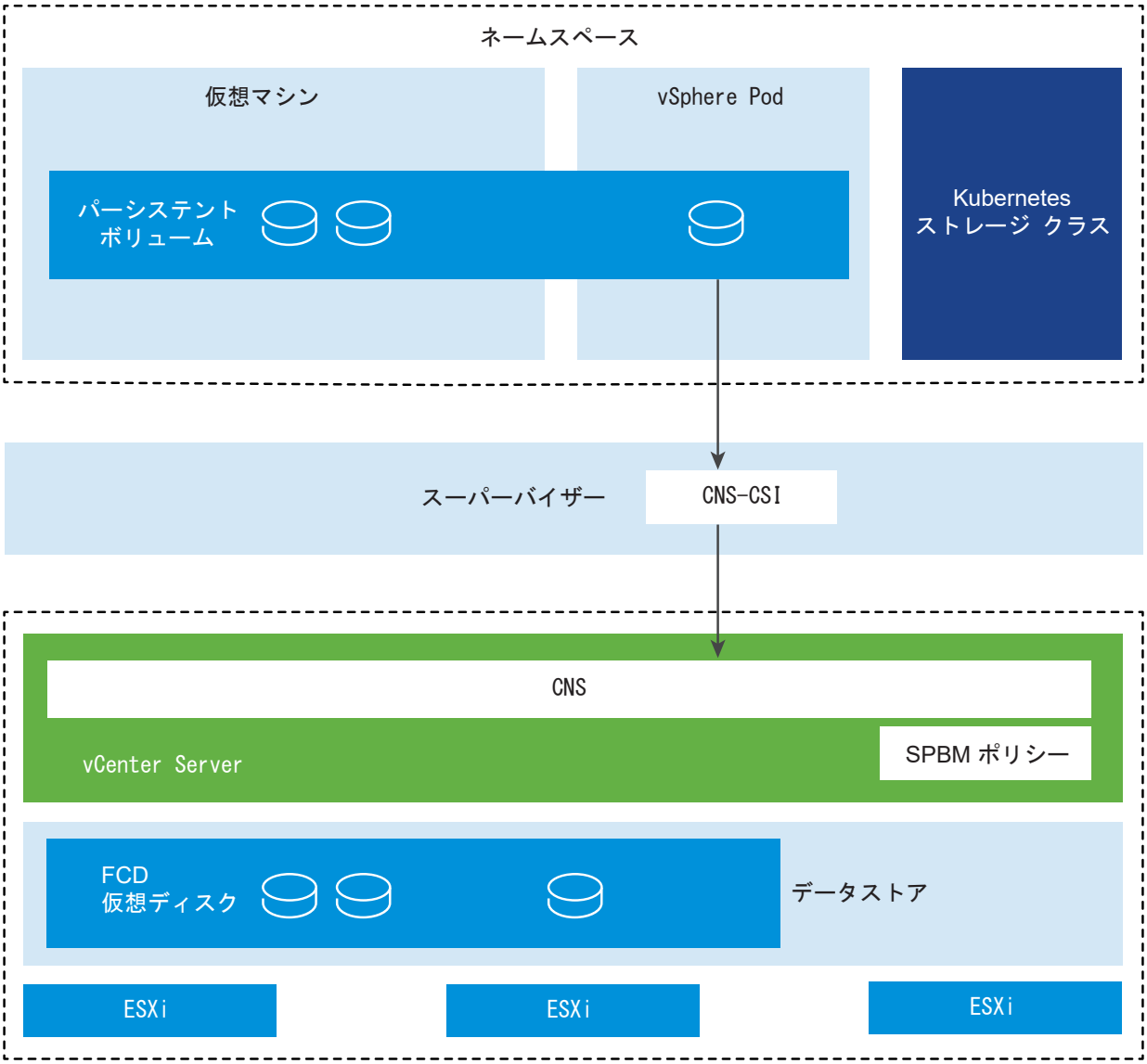
TKG サービス クラスタの vSphere 名前空間 ストレージ

vSphere クラウド ネイティブ ストレージ (CNS) は、Kubernetes ワークロードで使用するパーシステント ボリュームとそのバッキング仮想ディスクのプロビジョニングをサポートするストレージ ポリシーを提供します。

コンテナ ストレージ インターフェイス (CSI) は、コンテナのパーシステント ストレージをプロビジョニングするために Kubernetes が使用する業界標準です。スーパーバイザー は、vSphere 名前空間 を介して vSphere CNS ストレージを Kubernetes 環境に接続する CNS-CSI ドライバを実行します。vSphere CNS-CSI は、vSphere 名前空間 の TKG クラスタからのすべてのストレージ プロビジョニング要求について、CNS 制御プレーンと直接通信します。

TKG クラスタは、TKG クラスタから送信されるすべてのストレージ関連の要求を処理する vSphere CNS-CSI ドライバの変更バージョンを実行します。要求は スーパーバイザー の CNS-CSI に配信され、vCenter Server の CNS に伝達されます。

この図は、vSphere 名前空間、スーパーバイザー、および TKG クラスタ ストレージ メカニズム間の関係を示しています。



TKG サービス クラスタのパーシステント ストレージ ボリューム

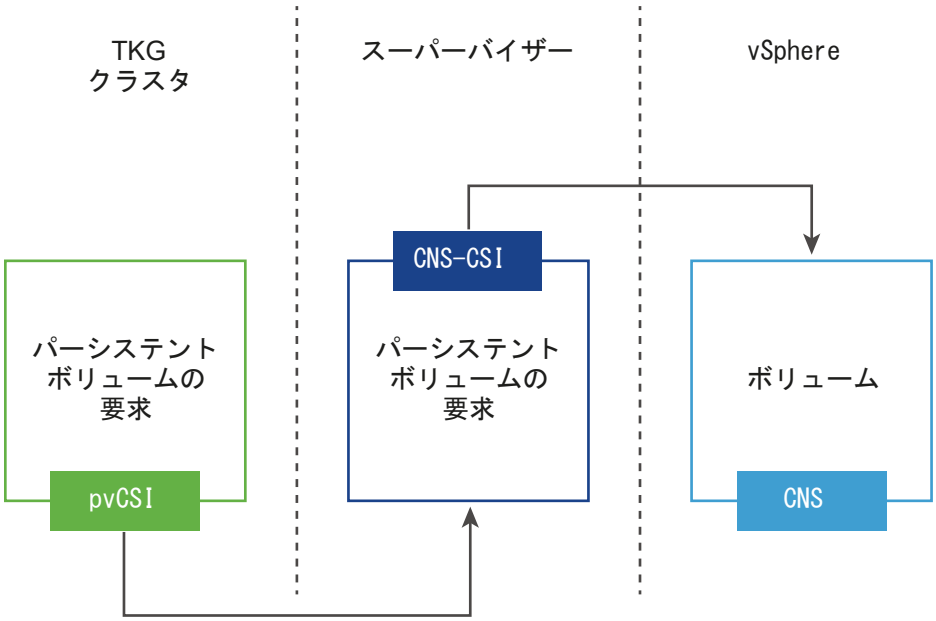
Kubernetes のステートフル アプリケーションには、パーシステント ボリュームが必要です。パーシステント ボリュームの詳細については、Kubernetes のドキュメントを参照してください。

vSphere 環境では、ステント ボリューム オブジェクトは、データストアにある仮想ディスクによってバッキングされます。データストアはストレージ ポリシーによって表されます。vSphere ストレージ ポリシーを vSphere 名前空間 に割り当てると、そのストレージ ポリシーは、その名前空間内の各 TKG クラスタの Kubernetes ストレージ クラスとして使用できるようになります。

TKG は、パーシステント ボリュームの動的プロビジョニングと静的プロビジョニングをサポートしています。動的プロビジョニングでは、パーシステント ボリュームを事前にプロビジョニングする必要はありません。vSphere 名前空間 で使用可能なストレージ クラスを参照するパーシステント ボリューム要求 (PVC) を発行してください。TKG は、対応するパーシステント ボリュームとバッキング仮想ディスクを自動的にプロビジョニングします。パーシステント ストレージ ボリュームの動的な作成を参照してください。

静的プロビジョニングでは、既存のストレージ オブジェクトを使用して、クラスタで使用できるようにします。パーシステント ボリュームを定義するには、既存のストレージ オブジェクト、それらのサポートされている構成、およびマウント オプションの詳細を指定する必要があります。パーシステント ストレージ ボリュームの静的な作成を参照してください。

この図は、動的なパーシステント ボリュームのプロビジョニング ワークフローを示しています。TKG クラスタで `kubectl` を使用して PVC を作成します。このアクションにより、スーパーバイザー で一致する PVC が生成され、CNS-CSI ドライバがトリガされ、CNS ボリューム作成 API が呼び出されます。



TKG サービス クラスタのストレージ クラス エディション

vSphere 名前空間 を構成するには、1 つ以上の vSphere ストレージ ポリシーを割り当てます。vSphere ストレージ ポリシーが適用されると、Kubernetes ストレージ クラスに変換され、スーパーバイザー にレプリケートされます。同様に、TKG コントローラは、その vSphere 名前空間 にデプロイされた各 TKG クラスタにストレージ クラスをレプリケートします。

TKG クラスタ側には、ストレージ クラスの 2 つのエディションが表示されます。1 つは、vSphere ストレージ ポリシーの作成時に指定されたユーザー定義の名前、もう 1 つは、`*-latebinding` が追加された別の名前です。

TKG ポッド スケジューラによってコンピューティング ノードが選択されたら、開発者はストレージ クラスの遅延バインド エディションを使用して、パーシステント ストレージ ボリュームにバインドできます。使用するストレージ クラスとタイミングの詳細については、パーシステント ボリュームのストレージ クラスの使用を参照してください。

```
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 名前空間 の作成

vSphere 名前空間 を作成する方法はいくつかあります。

管理者は、vSphere Client を使用して vSphere 名前空間 を作成できます。TKG サービス クラスタをホストするための vSphere 名前空間 の作成を参照してください。

vSphere 名前空間 の [所有者] ロール権限が付与されている vCenter Single Sign-On ユーザーは、`kubectl` を使用してセルフサービス方式で vSphere 名前空間 を作成できます。Kubectl を使用した vSphere 名前空間 の作成の有効化を参照してください。

VMware は vSphere 名前空間 のライフサイクルを管理するための vCenter Server API を公開し、以下を含むソフトウェア開発キット (SDK) を提供しています。

- 名前空間管理 API は、vSphere 名前空間 を管理するための REST ベースのリソースを提供します。
- 名前空間 API は、vSphere 名前空間 上のサブジェクトのアクセス コントロールを管理するための REST ベースのリソースを提供します。
- vSphere Automation SDK for Java には、vSphere 名前空間 の作成とライフサイクル管理を自動化するためのいくつかのパッケージが用意されています。
- vSphere Automation SDK for Python には、vSphere 名前空間 の作成とライフサイクル管理を自動化するためのいくつかのパッケージが用意されています。

TKG サービス クラスタをホストするための vSphere 名前空間 の作成

vSphere 名前空間 で TKG サービス クラスタをプロビジョニングします。

1 つ以上の TKG サービス クラスタをホストする vSphere 名前空間 を作成します。

手順

1. vSphere Client を使用して、vCenter Server にログインします。
2. [ワークロード管理] - [名前空間] - [新規名前空間] の順に選択します。
3. スーパーバイザー が有効になっている vSphere クラスタを指定します。
4. vSphere 名前空間 の名前を入力します。

   名前は DNS 準拠の形式にする必要があり、vCenter Server によって管理されるすべての スーパーバイザー インスタンスを通じて一意である必要があります。
5. vSphere 名前空間 の説明を入力します。

6 スーパーバイザー に使用されるネットワーク スタックのタイプに基づいて、[名前空間ネットワーク] を指定します。

ネットワーク スタック	
vSphere のネットワーク	[ネットワーク] メニューからワークロード ネットワークを選択します。
NSX ネットワーク	スーパーバイザー からネットワークを継承します。この場合、何も操作する必要はありません。 または、[クラスタ ネットワーク設定のオーバーライド] を選択し、この vSphere 名前空間用にネットワークをカスタマイズします。vSphere 名前空間のワークロード ネットワーク設定のオーバーライドを参照してください。

7 [作成] をクリックして vSphere 名前空間 を作成します。

名前空間が スーパーバイザー に作成されます。

8 TKG クラスタの vSphere 名前空間 を構成します。TKG サービス クラスタ向けの vSphere 名前空間 の構成を参照してください。

TKG サービス クラスタ向けの vSphere 名前空間 の構成

1 つ以上の TKG サービス クラスタを vSphere 名前空間 にデプロイします。vSphere 名前空間 に適用される構成設定は、そこにデプロイされている各 TKG サービス クラスタによって継承されます。

vSphere 名前空間 のロール権限の構成

ロールの権限は、範囲が vSphere 名前空間 に設定されます。TKG クラスタのユーザーとグループに割り当てることができるロール権限は、[所有者]、[編集可能]、[表示可能] の 3 つです。次の表で各ロールについて説明します。詳細については、TKG サービス クラスタの ID およびアクセス権の管理についてを参照してください。

vCenter Single Sign-On を使用している場合、3 つのすべてのロールを使用できます。SSO ユーザーおよびグループを vSphere 名前空間 に割り当てるには、vCenter Single Sign-On ユーザーおよびグループに対する vSphere 名前空間 権限の構成を参照してください。

外部 OIDC プロバイダを使用している場合は、[所有者] ロール権限を使用できません。OIDC ユーザーおよびグループを vSphere 名前空間 に割り当てるには、外部 ID プロバイダのユーザーおよびグループに対する vSphere 名前空間 権限の構成を参照してください。

ロール	説明
所有者	ロールの権限とバインドが割り当てられたユーザーおよびグループは、kubectl を使用して vSphere 名前空間 オブジェクトを管理し、TKG クラスタを操作できます。Kubectl を使用した vSphere 名前空間 の作成の有効化を参照してください。
編集可能	ロールの権限とバインドが割り当てられたユーザーおよびグループは、vSphere 名前空間 オブジェクトを表示し、TKG クラスタを操作できます。 編集可能権限が付与された vCenter Single Sign-On ユーザー/グループは、その vSphere 名前空間 にデプロイされた各 TKG クラスタの Kubernetes cluster-admin ロールにバインドされます。
表示可能	ロールの権限とバインドが割り当てられたユーザーおよびグループは、vSphere 名前空間 オブジェクトを表示できます。 **注：** Kubernetes には、[表示可能] 権限をバインドできる同等の読み取り専用ロールがありません。Kubernetes ユーザーにクラスタへのアクセス権を付与する方法については、「開発者への TKG サービス クラスタへの vCenter SSO アクセス権付与」を参照してください。

vSphere 名前空間 のパーシステント ストレージの構成

1 つ以上の vSphere ストレージ ポリシーを vSphere 名前空間 に割り当てることができます。割り当てられたストレージ ポリシーによって、vSphere ストレージ環境でのパーシステント ボリュームのデータストア配置を制御します。

通常、vSphere 管理者が vSphere ストレージ ポリシーを定義します。vSphere Zones を使用している場合は、ストレージ ポリシーを `Zonal` トポロジで構成する必要があります。TKG サービス クラスタの vSphere ストレージ ポリシーの作成を参照してください。

vSphere ストレージ ポリシーを vSphere 名前空間 に割り当てるには、次の手順を実行します。

1 [ワークロード管理] - [名前空間] の順に選択してから、ターゲットの vSphere 名前空間 を選択します。

2 [ストレージ] タイルで [ストレージの追加] を選択します。

3 選択可能なオプションからストレージ ポリシーを 1 つ以上選択します。

vSphere 名前空間 に割り当てる vSphere ストレージ ポリシーごとに、一致する 2 つの Kubernetes ストレージ クラスがその vSphere 名前空間 に作成されます。これらのストレージ クラスは、その vSphere 名前空間 でデプロイされた各 TKG クラスタにレプリケートされます。パーシステント ボリュームのストレージ クラスの使用を参照してください。

vSphere 名前空間 の容量および使用量の制限の設定

vSphere 名前空間 を構成すると、vSphere 名前空間 のリソース プールが vCenter Server に作成されます。デフォルトでは、このリソース プールは容量と使用量の割り当てなしで構成されます。リソースはインフラストラクチャによって制限されます。

vSphere 名前空間 の [容量と使用量] タイルで、次の [制限] を構成できます。

CPU	vSphere 名前空間に対して予約する CPU リソースの量。
メモリ	vSphere 名前空間に対して予約するメモリの量。
ストレージ	vSphere 名前空間に対して予約するストレージ容量の合計。
ストレージ ポリシーの制限	vSphere 名前空間に関連付けた各ストレージ ポリシーに専用のストレージ容量を個別に設定します。

通常、TKG クラスタのデプロイに対し、リソース割り当てを vSphere 名前空間 に構成する必要はありません。割り当て制限を割り当てる場合、そこにデプロイされている TKG クラスタへの潜在的な影響を理解することが重要です。

CPU およびメモリの制限

TKG クラスタ ノードで保証型の仮想マシン クラス タイプが使用されている場合、vSphere 名前空間 に構成された CPU およびメモリの制限は、そこにデプロイされている TKG クラスタには影響しません。ただし、TKG クラスタ ノードで TKG サービス クラスタでの仮想マシン クラスの使用が使用されている場合、CPU とメモリの制限が TKG クラスタに影響する可能性があります。

ベスト エフォート型の仮想マシン クラス タイプではリソースをオーバーコミットできるため、TKG クラスタをプロビジョニングする vSphere 名前空間 に CPU とメモリの制限を設定していてもリソースを使い果たすおそれがあります。競合が発生し、TKG クラスタ制御プレーンが影響を受けると、クラスタの実行が停止する場合があります。そのため、本番クラスタには、必ず TKG サービス クラスタでの仮想マシン クラスの使用を使用してください。すべての本番ノードに保証型の仮想マシン クラス タイプを使用できない場合は、少なくとも制御プレーン ノードに使用してください。

ストレージおよびストレージ ポリシーの制限

vSphere 名前空間 に構成されているストレージ制限により、そこにデプロイされているすべての TKG クラスタに対し vSphere 名前空間 で利用可能なストレージの全体的な量が決まります。

vSphere 名前空間 に構成されているストレージ ポリシーの制限により、ストレージ クラスがレプリケートされる TKG クラスタごとに、そのストレージ クラスで利用可能なストレージ容量が決まります。

一部のワークロードには最小ストレージ要件があります。例については、#unique_112 を参照してください。

TKR コンテンツ ライブラリの TKG サービス への関連付け

TKG クラスタをプロビジョニングするには、TKG サービス をコンテンツ ライブラリに関連付けます。TKr イメージをホストするためにコンテンツ ライブラリを作成するには、「5 章 TKG サービス クラスタ用 Kubernetes リリースの管理」を参照してください。

TKr コンテンツ ライブラリを vSphere 名前空間 に関連付けるには、次の手順を実行します。

1 [ワークロード管理] - [スーパーバイザー] - [スーパーバイザー] の順に選択します（スーパーバイザー インスタンスを選択します）。

2 [構成] - [スーパーバイザー] - [全般] - [Tanzu Kubernetes Grid Service] の順に選択します。

3 [コンテンツ ライブラリ] - [編集] の順に選択します。

4 TKR コンテンツ ライブラリを選択します。

5 vSphere 名前空間 に移動して、[名前空間の管理] を選択します。

6 選択したコンテンツ ライブラリが [Tanzu Kubernetes Grid Service] の構成ペインに表示されていることを確認します。

TKR コンテンツ ライブラリは名前空間の範囲内にないことに注意してください。すべての vSphere 名前空間 では、Tanzu Kubernetes Grid Service (TKGS) 用に構成された同じ TKR コンテンツ ライブラリを使用します。TKGS 用の TKR コンテンツ ライブラリの編集は、各 vSphere 名前空間 に適用されます。

注： [仮想マシン サービス] タイルで参照されているコンテンツ ライブラリは、TKR コンテンツ ライブラリではなく、スタンドアローン仮想マシンで使用するためのものです。TKR コンテンツ ライブラリをこのタイルに追加しないでください。

仮想マシン クラスの vSphere 名前空間 への関連付け

vSphere IaaS control plane ではデフォルトの TKG サービス クラスタでの仮想マシン クラスの使用を複数提供しており、ユーザーは独自のクラスを作成できます。

TKG クラスタをプロビジョニングするには、1 つ以上の TKG サービス クラスタでの仮想マシン クラスの使用をターゲットの vSphere 名前空間 と関連付けます。バインドされたクラスは、その vSphere 名前空間 でデプロイされている TKG クラスタ ノードで使用できます。

デフォルトの仮想マシン クラスを vSphere 名前空間 に関連付けるには、vSphere Client を使用して vCenter Server にログインし、次の手順を実行します。

1 [ワークロード管理] - [名前空間] の順に選択してから、ターゲットの vSphere 名前空間 を選択します。

2 [仮想マシン サービス] タイルで [仮想マシン クラスの追加] をクリックします。

3 追加する各仮想マシン クラスを選択します。

   a デフォルトの仮想マシン クラスを追加するには、リストの 1 ページ目の表ヘッダー内のチェック ボックスをオンにして 2 ページ目に移動し、そのページの表ヘッダーのチェック ボックスをオンにします。すべてのクラスが選択されていることを確認します。

   b カスタム クラスを作成するには、[新しい VM クラスの作成] をクリックします。手順については、仮想マシン サービスのドキュメントを参照してください。

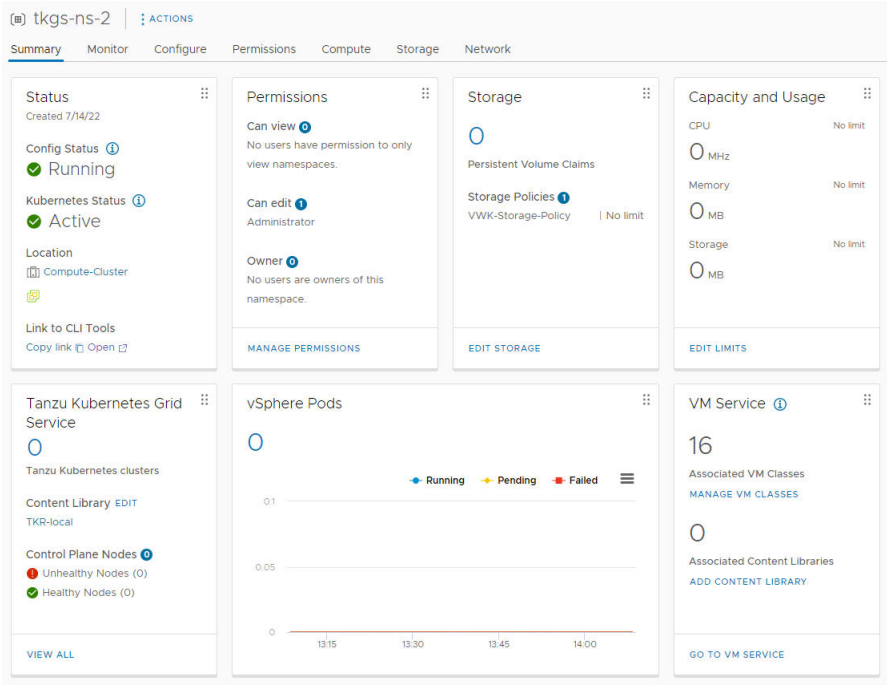
4 [OK] をクリックして操作を完了します。

5 クラスが追加されていることを確認します。[VM サービス] タイルに [VM クラスの管理] が表示されます。

vSphere 名前空間 構成の確認

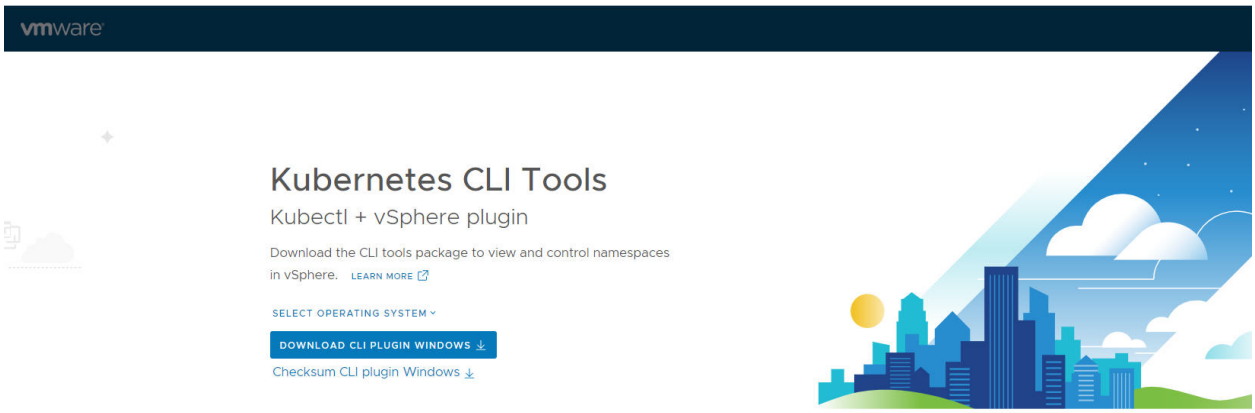
構成された vSphere 名前空間 には、[ステータス]、[権限]、[ストレージ]、[容量と使用量]、[TKR コンテンツ ライブラリ]、および [仮想マシン クラス] が含まれます。

図 6-1. 構成済み vSphere 名前空間



[ステータス] タイルには、vSphere IaaS control plane CLI ツールへのリンクが含まれています。DevOps 画面は、スーパーバイザー 制御プレーン ロード バランサによって提供されます。vSphere 向け Kubernetes CLI Tools をダウンロードするためのリンクを TKG クラスタ ユーザーに提供します。vSphere 向け Kubernetes CLI Tools のインストールを参照してください。

図 6-2. vSphere 名前空間 DevOps 画面



kubectl を使用した vSphere 名前空間 の構成を確認するには、TKG サービス クラスタをホストするための vSphere 名前空間 の準備の確認を参照してください。

vSphere 名前空間のワークロード ネットワーク設定のオーバーライド

スーパーバイザーをデプロイする場合は、ワークロード ネットワークのデフォルト設定を構成します。NSX ネットワークでスーパーバイザーを有効にした場合は、vSphere 名前空間の作成時にデフォルトのワークロード ネットワーク設定をオーバーライドできます。オーバーライドされたワークロード ネットワーク設定は、この vSphere 名前空間セグメントにのみ適用されます。

ワークロード ネットワーク設定のオーバーライド（NSX のみ）

vSphere 名前空間 を作成すると、ネットワーク セグメントが作成されます。デフォルトでは、このネットワーク セグメントは、スーパーバイザー で構成されたワークロード ネットワークから取得されます。詳細については vSphere 名前空間 ネットワークを参照してください。

スーパーバイザー に NSX ネットワークが構成されている場合、vSphere 名前空間 の作成時に、vSphere 名前空間 に対して [クラスタ ネットワーク設定のオーバーライド] を選択することができます。このオプションを選択すると、Ingress、Egress、および名前空間ネットワークのフィールドに CIDR を追加して、vSphere 名前空間 ネットワークをカスタマイズできます。新しい CIDR を追加すると、この vSphere 名前空間 インスタンスの既存の CIDR がオーバーライドされます。

NSX バージョン 4.1.1 以降を構成し、Enterprise ライセンスの NSX Advanced Load Balancer バージョン 22.1.4 以降を NSX にインストール、構成、および登録している場合、NSX で使用されるロード バランサは NSX Advanced Load Balancer です。4.1.1 よりも前のバージョンの NSX を構成している場合は、NSX ロード バランサが使用されます。詳細については、『vSphere IaaS 制御プレーンの概念と計画』の「スーパーバイザー ネットワーク」を参照してください。

スーパーバイザー ネットワーク設定をオーバーライドする一般的な使用事例では、ルーティング可能なポッド ネットワークを使用して TKG クラスタをプロビジョニングします。これを行う方法の詳細および例のリンクについては、表の構成設定を参照してください。

表 6-1. vSphere 名前空間 ネットワーク プランニングに関する考慮事項

検討事項	説明
必要な NSX	特定の vSphere 名前空間 の スーパーバイザー ネットワーク設定をオーバーライドするには、スーパーバイザー に NSX ネットワークを構成する必要があります。
NSX のインストール	特定の vSphere 名前空間 の スーパーバイザー ネットワーク設定をオーバーライドするには、NSX のインストールに、Tier-0 ゲートウェイ（ルーター）専用の Edge クラスタと、Tier-1 ゲートウェイ専用の別の Edge クラスタを含める必要があります。『vSphere with Tanzu のインストールと構成』ガイドに記載された NSX インストール手順を参照してください。

表 6-1. vSphere 名前空間 ネットワーク プランニングに関する考慮事項 （続き）

検討事項	説明
必要な IP アドレス管理	特定の vSphere 名前空間 の スーパーバイザー ネットワーク設定をオーバーライドする場合、新しい vSphere 名前空間 ネットワークでは、スーパーバイザー および他の vSphere 名前空間 ネットワークから一意の Ingress サブネット、Egress サブネット、および名前空間ネットワーク サブネットを指定する必要があります。設定に応じて、IP アドレスの割り当てを管理する必要があります。
スーパーバイザーのルーティング	スーパーバイザー は、TKG クラスタ ノードと Ingress サブネットに直接ルーティングできる必要があります。vSphere 名前空間 に Tier-0 ゲートウェイを選択する場合、必要なルーティングを構成する方法は 2 つあります。 ■ Virtual Routing and Forwarding (VRF) ゲートウェイを使用して、スーパーバイザー Tier-0 ゲートウェイから構成を継承する ■ Border Gateway Protocol (BGP) を使用して、スーパーバイザー Tier-0 ゲートウェイと専用 Tier-0 ゲートウェイの間にルートを構成する これらのオプションの詳細については、NSX Tier-0 ゲートウェイのドキュメントを参照してください。

スーパーバイザー ネットワーク設定をオーバーライドするための構成フィールド。

表 6-2. ワークロード ネットワーク設定をオーバーライドするための vSphere 名前空間の構成オプション

コンポーネント	構成
Tier-0 ゲートウェイ	NSX Tier-0 ゲートウェイは、スーパーバイザー を物理ネットワークに接続します。選択した Tier-0 ゲートウェイは、vSphere 名前空間 用に作成された Tier-1 ゲートウェイに関連付けられます。 新しい Tier-0 ゲートウェイを選択すると、スーパーバイザー を有効にするときに構成された Tier-0 ゲートウェイがオーバーライドされます。この場合は、新しい CIDR 範囲を構成する必要があります。Tier-0 ゲートウェイにリンクされている VRF ゲートウェイを選択すると、ネットワークとサブネットが自動的に構成されます。 Tier-0 ゲートウェイを選択して構成を完了すると、Tier-0 ゲートウェイを変更できなくなります。
ロード バランサのサイズ	vSphere 名前空間 の Tier-1 ゲートウェイ上のロード バランサ インスタンスのサイズを選択します。 ロード バランサのサイズを [小]（デフォルト）、[中]、または [大] に設定します。Edge ノードごとに、設定された数のロード バランサ インスタンスのみを定義できます。詳細については、構成の上限を参照してください。 **注：** この設定は NSX Advanced Load Balancer には該当しません。
NAT モード	NAT モードは、デフォルトで選択されています。つまり、名前空間ネットワーク サブネットはルーティング不可になると想定されるため、名前空間ネットワーク、Ingress CIDR、および Egress CIDR を構成する必要があります。 NAT モードを選択解除することにより、ユーザーが名前空間ネットワークにルーティング可能な CIDR 範囲を指定する予定であることがシステムに通知されます。NAT モードを選択解除すると、Tier-0 ゲートウェイの外部から TKG クラスタ ノードの IP アドレスに直接アクセスできるようになります。Egress CIDR を構成する必要はありません。 NAT モードを使用せずにクラスタをプロビジョニングするには、NAT モードを選択解除します。7 章 TKG サービス クラスタのプロビジョニングの例を参照してください。

表 6-2. ワークロード ネットワーク設定をオーバーライドするための vSphere 名前空間の構成オプション （続き）

コンポーネント	構成
名前空間ネットワーク CIDR	名前空間ネットワーク CIDR は IP アドレス プールとして動作するサブネットであり、名前空間サブネット プリフィックスには、その IP アドレス プールから分割された後続の CIDR ブロックのサイズが記述されます。 vSphere 名前空間 が作成されるたびに、名前空間ネットワークのサブネットが割り当てられます。このブロックから分割されたサブネット サイズは /24 です。つまり、vSphere 名前空間 ごとに最大 256 ポッドを作成できます。詳細については、「構成の上限」を参照してください。 vSphere 名前空間 セグメントに接続された TKG クラスタに IP アドレスを割り当てるには、名前空間ネットワークの CIDR を使用します。 NAT モードが選択されている場合、CIDR はルーティング不可であると想定されます。NAT モードがオフの場合、名前空間ネットワーク CIDR はルーティング可能である必要があります。
名前空間サブネット プリフィックス	名前空間セグメント用に予約されるサブネットのサイズを指定する、サブネット プリフィックスを入力します。デフォルトは 28 です。 名前空間サブネット プリフィックスは、vSphere 名前空間 セグメントごとに作成される IP サブネットを定義します。たとえば、プリフィックスとして /24 を設定すると、254 個の IP アドレスを含む IP サブネットが設定された vSphere 名前空間 セグメントが生成され、そこにデプロイされた TKG クラスタには、これらの IP アドレスが割り当てられます。 その他の例： 名前空間ネットワーク CIDR = 192.168.1.0/24 名前空間サブネット プリフィックス = /28 この場合、TKGS では 16 個の 192.168.1.x/28 CIDR ブロックを 192.168.1.0/24 サブネットから提供できます。これにより、TKG サービス の管理対象仮想マシン（TKC、仮想マシン、vSphere Pod）が接続されている 16 個の TKGS 名前空間ネットワークをインスタンス化できます。たとえば、すべての TKC は専用の名前空間 CIDR（ここでは 192.168.1.0/28）を受信し、次の TKC 名前空間サブネットは 192.168.0.16/28 のようになります。
Ingress	Ingress IP アドレス CIDR ブロックは、サービス タイプのロード バランサとすべての vSphere 名前空間 の Ingress コントローラで公開されている Kubernetes サービスに IP アドレスを割り当てる際に使用されます。TKG クラスタ サービスと Ingress は、この CIDR ブロックから IP アドレスを取得します。 TKG クラスタのロード バランサ サービスによって公開される仮想 IP アドレスの Ingress IP アドレス範囲を決定する CIDR 注釈を入力します。 **注：** この設定は NSX Advanced Load Balancer には該当しません。
Egress	Egress IP アドレス CIDR は、vSphere 名前空間 から送信されて外部サービスにアクセスするトラフィック用の SNAT（送信元ネットワーク アドレス変換）で IP アドレスを割り当てる際に使用されます。 SNAT IP アドレスの Egress IP アドレス範囲を決定する CIDR 注釈を入力します。

TKG サービス クラスタでの仮想マシン クラスの使用

TKG サービス クラスタ ノードのサイズを設定するには、仮想マシン (VM) クラスを指定します。このプラットフォームにはデフォルトの仮想マシン クラスが用意されていますが、ユーザーが独自のクラスを作成できます。仮想マシン クラスを使用するには、そのクラスをターゲット vSphere 名前空間 に関連付けて、クラスタ マニフェストでクラスを参照します。

仮想マシン クラスについて

仮想マシン (VM) クラスは、CPU やメモリ (RAM) などの仮想マシン (VM) の処理能力を確保するためのリソース予約の要求です。たとえば、「guaranteed-large」という名前の仮想マシン クラス タイプは、4 個の CPU と 16 GB の RAM を予約します。

注： 仮想マシンのディスク サイズは、仮想マシン クラスの定義ではなく、OVA テンプレートによって設定されます。Tanzu Kubernetes リリース の場合、ディスク サイズは 16 GB になります。

仮想マシン クラスには、保証型とベスト エフォート型という 2 つの予約タイプがあります。保証型のクラスは、構成したリソースをすべて予約します。つまり、指定したクラスタで `spec.policies.resources.requests` と `spec.hardware` の設定が一致します。ベスト エフォート型のクラスでは、リソースをオーバーコミットできます。本番環境のワークロードでは、保証型の仮想マシン クラス タイプの使用をお勧めします。

注意: ベスト エフォート型の仮想マシン クラス タイプではリソースをオーバーコミットできるため、TKG クラスタをデプロイする vSphere 名前空間 に制限を設定していてもリソースを使い果たす可能性があります。競合が発生し、制御プレーンが影響を受けると、クラスタの実行が停止する場合があります。そのため本番クラスタには保証型の仮想マシン クラス タイプを使用します。すべての本番ノードに保証型の仮想マシン クラス タイプを使用できない場合は、少なくとも制御プレーン ノードに使用してください。

TKG サービス クラスタでの仮想マシン クラスの使用

TKG サービス クラスタで仮想マシン クラスを使用するには、このクラスタがプロビジョニングされている vSphere 名前空間 に仮想マシン クラスをバインドする必要があります。この操作を行うには、仮想マシン クラスをターゲット名前空間に関連付けます。TKG サービス クラスタ向けの vSphere 名前空間 の構成を参照してください。

ターゲット vSphere 名前空間 で使用可能な仮想マシン クラスを一覧表示するには、`kubectl get virtualmachineclass` コマンドを使用します。

注： このコマンドで問題が発生した場合は、「仮想マシン クラスのエラーのトラブルシューティング」を参照してください。

仮想マシン クラスの定義は変更できません。すべての仮想マシンは、デフォルトの仮想マシン クラス定義を含めて、編集することができます。仮想マシン クラスを編集した場合、既存の TKG クラスタ ノードは影響を受けません。新しい TKG クラスタでは、変更されたクラスが使用されます。

注意： TKG クラスタで使用されている仮想マシン クラスを編集し、そのクラスタをスケール アウトした場合、新しいノードは編集されたクラス定義を使用しますが、既存のノードは最初のクラス定義を使用するため、クラスの不一致が発生します。

デフォルトの仮想マシン クラス

表には、Tanzu Kubernetes クラスタ ノードのデプロイ サイズとして使用される、デフォルトの仮想マシン クラスのタイプが一覧表示されています。

リソースのオーバーコミットを回避するには、本番ワークロードで保証されたクラス タイプを使用する必要があります。環境（開発、テスト、本番）にワークロードをデプロイしている場合にメモリ不足を回避するには、ワーカー ノードに小さなクラス サイズまたは極端に小さなクラス サイズを使用しないでください。

表 6-3. デフォルトの仮想マシン クラス

クラス	CPU	メモリ (GB)	予約済みの CPU とメモリ
guaranteed-8xlarge	32	128	はい
best-effort-8xlarge	32	128	なし
guaranteed-4xlarge	16	128	はい
best-effort-4xlarge	16	128	なし
guaranteed-2xlarge	8	64	はい
best-effort-2xlarge	8	64	なし
guaranteed-xlarge	4	32	はい
best-effort-xlarge	4	32	なし
guaranteed-large	4	16	はい
best-effort-large	4	16	なし
guaranteed-medium	2	8	はい
best-effort-medium	2	8	なし
guaranteed-small	2	4	はい
best-effort-small	2	4	なし
guaranteed-xsmall	2	2	はい
best-effort-xsmall	2	2	なし

カスタムの仮想マシン クラス

vSphere IaaS control plane は、TKG サービス クラスタで使用するカスタム仮想マシン クラスをサポートします。カスタム仮想マシン クラスを定義したら、クラスタで使用する前に、ターゲット vSphere 名前空間 に関連付ける必要があります。詳細については、仮想マシン サービスのドキュメントを参照してください。

TKG サービス クラスタをホストするための vSphere 名前空間 の準備の確認

vSphere 名前空間 を構成したら、スーパーバイザー にログインし、vSphere 名前空間 に TKG サービス クラスタをホストするする準備ができていることを確認します。

TKG サービス クラスタのプロビジョニングのための vSphere 名前空間 構成の確認

TKG サービス クラスタのプロビジョニングを準備するには、このタスクを完了して、vSphere 名前空間 が適切に構成されていることを確認します。

1 スーパーバイザー にログインします。

```
kubectl vsphere login --server IP-ADDRESS-SUPERVISOR-CLUSTER --vsphere-username VCENTER-SSO-USERNAME
```

2 1 つ以上の TKG クラスタをプロビジョニングする、対象の vSphere 名前空間 にコンテキストを切り替えます。

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

3 vSphere 名前空間 を指定します。

```
kubectl describe ns VSPHERE-NAMESPACE-NAME
```

このコマンドによって、vSphere 名前空間 で使用できる各ストレージ クラスの名前とリソース割り当てが返されます。

4 使用可能な Tanzu Kubernetes リリース を一覧表示して、説明を記述します。

```
kubectl get tanzukubernetesreleases
```

このコマンドによって、vSphere 名前空間 用に構成されたコンテンツ ライブラリにあり、そのライブラリに同期またはアップロードされている TKR が返されます。

5 使用可能な仮想マシン クラスを一覧表示します。

```
kubectl get virtualmachineclass
```

このコマンドによって、名前空間に関連付けられている仮想マシン クラスが返されます。TKG サービス クラスタ ノードのプロビジョニングに使用できるのは、バインドされた仮想マシン クラスのみです。

Kubectl を使用した vSphere 名前空間 の作成の有効化

vSphere 名前空間 サービスを有効にすると、開発者は `kubectl` を使用して vSphere 名前空間 のライフサイクルを管理できるようになります。

スーパーバイザー で vSphere 名前空間 サービスを有効にすると、vSphere 名前空間 の所有者ロールに割り当てられた開発者は、`kubectl create namespace <NAME>` コマンドを使用して独自の vSphere 名前空間 を作成できます。

vSphere 名前空間 サービスを有効にするときは、名前空間テンプレートを定義して有効にします。所有者ロールに割り当てられた開発者が、テンプレートを使用して名前空間を作成します。

手順

1 vSphere Client を使用して、vCenter Server にログインします。

2 [ワークロード管理] - [スーパーバイザー] の順に選択し、スーパーバイザー インスタンスを選択します。

3 [構成] タブで、[スーパーバイザー] - [全般] の順に選択します。

4 [名前空間サービス] を選択します。

5 [ステータス] スイッチを切り替えて、この機能を有効にします。

[名前空間テンプレートの作成] 画面が表示されます。

6 [構成] ペインで、名前空間に対するリソース制限を構成します。

オプション	説明
CPU	名前空間に予約する CPU リソースの量。
メモリ	名前空間に予約するメモリの量。
ストレージ	名前空間に予約するストレージ容量の合計。
ストレージ ポリシー	名前空間に関連付けた各ストレージ ポリシーに専用のストレージ容量を個別に設定します。
仮想マシン クラス	仮想マシン クラスを選択します。複数を選択するには Ctrl キーを使用します。
コンテンツ ライブラリ	TKR コンテンツ ライブラリを選択します。

7 [次へ] をクリックします。

8 [権限] ペインで DevOps エンジニアとグループを追加して、テンプレートを使用して名前空間を作成できるようにします。

a vsphere.local ID ソースを選択します（vSphere SSO を使用する必要があります）。

b ユーザーまたはグループを選択します。

c 所有者ロールを選択します。

9 [確認] ペインに、構成するプロパティが表示されます。

プロパティを確認して、[完了] をクリックします。

10 vSphere 名前空間 サービスが有効になっていることを確認します。

vSphere 名前空間 テンプレートが構成され、アクティブな状態になっています。所有者ロールに割り当てられている vSphere 名前空間 ユーザー/グループはテンプレートを使用し、kubectl コマンド `kubectl create namespace <NAME>` を使用して vSphere 名前空間 を作成できます。

vSphere 名前空間 の削除

スーパーバイザー から vSphere 名前空間 を削除できます。その前に、プロビジョニングされている TKG サービス クラスタがあれば、それを削除する必要があります。

前提条件：vSphere 名前空間内の TKG サービス クラスタの削除

vSphere 名前空間 を削除する前に、プロビジョニングされているすべての TKG 2.0 クラスタを削除する必要があります。

注： また、vSphere 名前空間 にデプロイされている vSphere ポッド もすべて削除する必要があります。

TKG クラスタを削除するには、kubectl または Tanzu CLI を使用します。Kubectl または Tanzu CLI を使用した TKG クラスタの削除を参照してください。

注： vSphere Client または vCenter Server CLI を使用して TKG クラスタの削除を試みないでください。

vSphere 名前空間の削除

TKG 2.0 クラスタのプロビジョニングを準備するには、このタスクを完了して、vSphere 名前空間 が適切に構成されていることを確認します。

1 vSphere Client を使用して、vCenter Server にログインします。

2 ワークロード管理 を選択します。

3 [名前空間] タブを選択します。

スーパーバイザー 上に作成された各 vSphere 名前空間 が一覧表示されます。

4 削除する vSphere 名前空間 を選択します。

5 [削除] を選択します。

vSphere 名前空間 が削除されます。この処理は完了するまでに時間がかかる場合があります。進行状況を確認するには、タスク ペインを使用してください。

TKG サービス クラスタのプロビジョニング

7

TKG サービスを使用して、Tanzu Kubernetes クラスタと、ClusterClass に基づくクラスタの、2 種類のワークロード クラスタをプロビジョニングできます。

次のトピックを参照してください。

- TKG クラスタのプロビジョニングについて
- Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフロー
- Tanzu CLI を使用して TKG クラスタをプロビジョニングするためのワークフロー
- Kubectl を使用した TKG クラスタ プロビジョニングのテスト
- Kubectl または Tanzu CLI を使用した TKG クラスタの削除
- クラスタ v1beta1 API の使用
- TanzuKubernetesCluster v1alpha3 API の使用

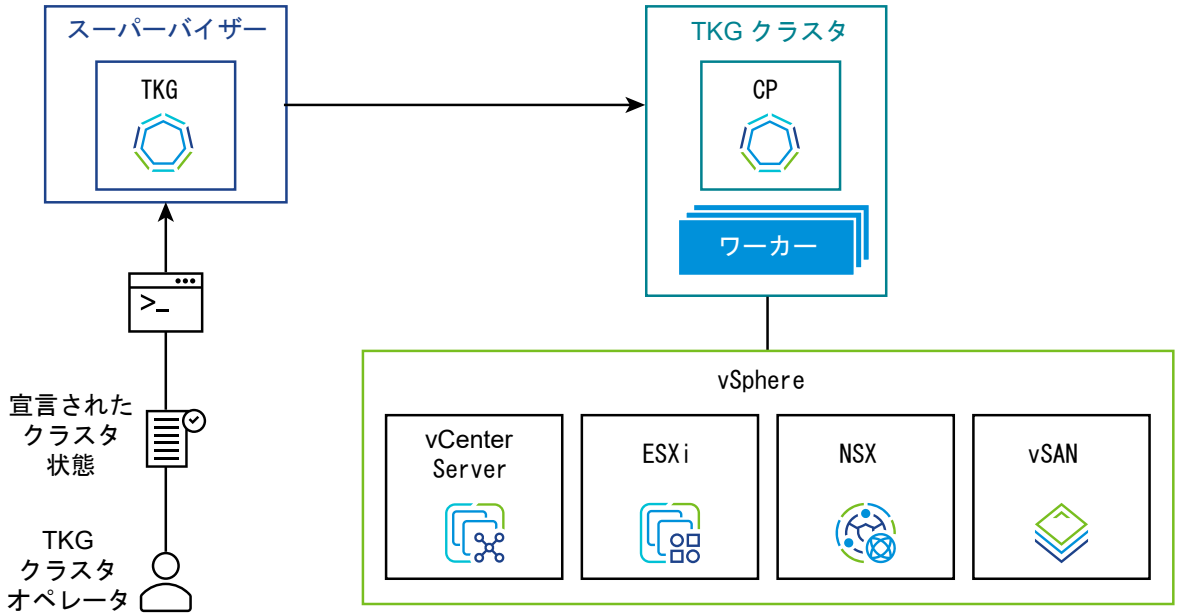
TKG クラスタのプロビジョニングについて

TKG サービスには、TKG クラスタのプロビジョニングとライフサイクルの管理を行うための 2 つの API とサポート クライアントがあります。

TKG クラスタのプロビジョニング

次の図は、スーパーバイザー に TKG クラスタをプロビジョニングするためのワークフローを示しています。

図 7-1. スーパーバイザー への TKG クラスタのプロビジョニング



TKG クラスタのタイプ

スーパーバイザー が管理クラスタとして動作し、Kubernetes Cluster API (CAPI) をホストする vSphere IaaS control plane インフラストラクチャには、2 つのタイプの Kubernetes ワークロード クラスタをプロビジョニングできます。各タイプは、ClusterClass に基づいています。サポートされているバージョンについては、TKR リリース ノートを参照してください。TKG サービス クラスタでの Kubernetes リリースの使用も参照してください。

tanzukubernetescluster という名前のデフォルトの ClusterClass を参照する、CAPI クラスタを使用する TanzuKubernetesCluster

クラスタ署名：

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
```

このタイプのワークロード クラスタは、`tanzukubernetescluster` という名前のデフォルトの ClusterClass を参照するクラスタ API (CAPI) を使用して構築された Tanzu Kubernetes クラスタです。このクラスタ タイプは [TanzuKubernetesCluster]、プロビジョニング API は `v1alpha3` です。これは CAPI クラスタ上での抽象化であるため、バックエンド クラスタ クラスへの参照はクラスタ マニフェストで指定されていません。参照はシステムによって処理されます。

このタイプのワークロード クラスタを使用する場合は、TanzuKubernetesCluster オブジェクトがフロントラインになり、抽象化レイヤーとして機能します。このタイプのクラスタをプロビジョニングするワークフローに、vSphere IaaS control plane バージョン 7 で TKGS クラスタをプロビジョニングする方法からの変更点はありません。

tanzukubernetescluster という名前のデフォルトの ClusterClass を参照する CAPI クラスタ

クラスタ署名：

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
```

このタイプのワークロード クラスタは、`tanzukubernetescluster` という名前のデフォルトの ClusterClass を参照するクラスタ API を使用して構築された純粋な CAPI クラスタです。このクラスタ タイプは [クラスタ]、プロビジョニング API は `v1beta1` です。

このタイプのワークロード クラスタを使用する場合は、CAPI クラスタ API がフロント ラインになり、TKC 抽象化レイヤーは存在しません。システムではインフラストラクチャを処理するためのコントローラが提供されているため、ClusterClass 関連のオブジェクトを作成する必要はありません。変数が公開されるため、クラスタをカスタマイズできます。クラスタ仕様内のフィールドは TKC 仕様内のフィールドとは異なりますが、プロビジョニング ワークフローは同じです。

TKG クラスタのプロビジョニング API

vSphere 8 スーパーバイザー 上の TKG は、TKG クラスタのライフサイクルを管理するための 2 つの API、`v1alpha3` と `v1beta1` を提供します。いずれの API も、Kubernetes API と同様に宣言型です。宣言型クラスタ プロビジョニングでは、TKG クラスタの目的の状態（ノードの数、使用可能なストレージ、仮想マシンのサイズ、Kubernetes ソフトウェアのバージョン）を指定します。TKG は、宣言された状態のクラスタをプロビジョニングおよび維持するための作業を実行します。

vSphere 8 スーパーバイザー で既存の Tanzu Kubernetes クラスタを TKG にアップグレードする場合は、アップグレード プロセスを開始する前に、そのクラスタが `v1alpha2` API を使用している必要があります。詳細については、アップグレードのドキュメント「#unique_51」を参照してください。

API	種類	vCenter Server のバージョン	説明
v1beta1	Cluster	vCenter Server 8 以降	クラスタ クラスに基づいてクラスタのライフサイクルを管理するための新しい API。
v1alpha3	TanzuKubernetesCluster	vCenter Server 8 以降	v1alpha2 API の後継。API v1alpha2 でサポートされていたすべての機能は、v1alpha3 API でもサポートされます。新しい機能も追加されました。
v1alpha2	TanzuKubernetesCluster	vCenter Server 7 U3	vCenter Server 7 U3 スーパーバイザーでの Tanzu Kubernetes クラスタのプロビジョニング、およびクラスタの vCenter Server 8 スーパーバイザーへのアップグレード用レガシー API。vSphere 8 へのアップグレード時またはプロビジョニング時に、v1alpha2 API は v1alpha3 API に自動変換されます。
v1alpha1	TanzuKubernetesCluster	vCenter Server 7 U1、U2	vCenter Server 7 スーパーバイザーの第 1 世代で Tanzu Kubernetes クラスタをプロビジョニングするための廃止済み API。

TKG クラスタのプロビジョニングのクライアント

vSphere 8 スーパーバイザー 上の TKG では、TKG クラスタをプロビジョニングするためのさまざまなクライアント ワークフローがサポートされます。

- Kubectl + YAML（Kubernetes 形式の宣言型クラスタ プロビジョニング用）。Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。
- Tanzu CLI（インタラクティブなコマンドライン クラスタ プロビジョニング用）。Tanzu CLI を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。
- Tanzu Mission Control（Web ベースのクラスタ プロビジョニング用）。ホスト型 Tanzu Mission Control の スーパーバイザー への登録を参照してください。

Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフロー

このワークフローに従い、kubectl コマンドと YAML に定義されたクラスタ仕様を使用して、TKG サービス クラスタを宣言によってプロビジョニングします。

このワークフローでは、kubectl と YAML を使用して、TKG クラスタを宣言によってプロビジョニングします。

前提条件

プロビジョニング ワークフローを開始する前に、次の前提条件を確認し、必要な手順を実施します。

- 使用環境に最新の スーパーバイザー バージョンをインストールするか、最新バージョンにアップデートします。2 章 TKG サービス クラスタの実行を参照してください。
- コンテンツ ライブラリを最新の Tanzu Kubernetes リリース で作成またはアップデートします。5 章 TKG サービス クラスタ用 Kubernetes リリースの管理を参照してください。
- TKG クラスタをホストするための vSphere 名前空間 を作成して構成します。6 章 TKG サービス クラスタをホストするための vSphere 名前空間 の構成を参照してください。

手順

1 vSphere 向け Kubernetes CLI Tools をインストールします。

vSphere 向け Kubernetes CLI Tools のインストール。

2 kubectl を使用して、スーパーバイザー で認証します。

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-username USERNAME
```

注： スーパーバイザー の FQDN は、有効になっている場合にのみ使用できます。詳細については、スーパーバイザー の構成に関するドキュメントを参照してください。

3 スーパーバイザー にログインできたことを確認します。

次のようなメッセージが表示されます。

```
Logged in successfully.

You have access to the following contexts:
   192.197.2.65
   tkg2-cluster-namespace
```

ここで、`192.197.2.65` は スーパーバイザー のコンテキストで、`tkg2-cluster-namespace` は TKG クラスタをプロビジョニングする vSphere 名前空間 のコンテキストです。

4 ターゲット vSphere 名前空間 が現在のコンテキストであることを確認します。

```
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
```

ターゲット vSphere 名前空間 が現在のコンテキストでない場合は、そのコンテキストに切り替えます。

```
kubectl config use-context tkg2-cluster-namespace
```

5 vSphere 名前空間 で使用可能な仮想マシン クラスを一覧表示します。

```
kubectl get virtualmachineclass
```

使用できるのは、ターゲット名前空間にバインドされている仮想マシン クラスのみです。仮想マシン クラスが表示されない場合は、デフォルトの仮想マシン クラスが vSphere 名前空間 に関連付けられていることを確認してください。仮想マシン クラスのエラーのトラブルシューティングも参照してください。

6 使用可能なパーシステント ボリューム ストレージ クラスを取得します。

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

このコマンドは、名前空間に関する詳細情報を返します。これには、`tkg2-storage-policy.storageclass.storage.k8s.io/requests.storage` のような形式でストレージ クラスの情報が含まれます。この文字列の最初のトークンはストレージ クラス名で、この例では `tkg2-storage-policy` となっています。`kubectl describe storageclasses` コマンドを実行すると使用可能なストレージ クラスも返されますが、それには vSphere 管理者の権限が必要です。

7 使用可能な Tanzu Kubernetes リリース を一覧表示します。

この操作を実行には、次のいずれかのコマンドを使用します。

```
kubectl get tkr
```

```
kubectl get tanzukubernetesreleases
```

このコマンドを使用すると、この vSphere 名前空間 で使用できる TKR が返され、デプロイ先の スーパーバイザー との互換性を確認できます。このコマンドで返されたリリースのみを使用できます。リリースが一切表示されない、または必要なリリースが表示されない場合は、次の手順を実行したことを確認してください。a) TKR コンテンツ ライブラリを作成する、b) コンテンツ ライブラリを目的の OVA ファイルと同期する、c) コンテンツ ライブラリを TKG クラスタをプロビジョニングしている vSphere 名前空間 に関連付ける。

8 TKG クラスタをプロビジョニングするための YAML ファイルを作成します。

a 作成するクラスタのタイプを決定して、その API と機能を確認します。

- TanzuKubernetesCluster：TanzuKubernetesCluster v1alpha3 API の使用
- クラスタ: クラスタ v1beta1 API の使用

b 最初に、クラスタをプロビジョニングするためのサンプル YAML を 1 つ用意します。

例：

- v1alpha3 の例：デフォルトの TanzuKubernetesCluster
- v1beta1 の例：デフォルト クラスタ

c YAML ファイルを `tkg2-cluster-1.yaml` などの名前で保存します。

d 前述のコマンドの出力によって得られた情報を使用して、YAML に次の情報を入力します。

- `tkg2-cluster-1` などのクラスタの名前
- 対象となる vSphere 名前空間
- `guaranteed-medium` などのバインドされた仮想マシン クラス
- クラスタ ノードとパーシステント ボリュームのストレージ クラス
- 制御プレーンおよびワーカー ノード（レプリカ）の数
- `v1.25.7+vmware.3-fips.1-tkg.1` などの TKR NAME 文字列で指定された Tanzu Kubernetes リリース

e 必要に応じて TGK クラスタ YAML をカスタマイズします。

- containerd などの変更の多いコンポーネント用のボリュームを個別に追加します。
- クラスタ ノードとパーシステント ボリュームのデフォルトのパーシステント ストレージ クラスを指定します。
- CNI、ポッド、サービス CIDR などのクラスタ ネットワークをカスタマイズします。

f YAML 構文チェッカーを使用し、YAML が有効であることを確認します。

この手順を行うと、TKG クラスタをプロビジョニングするための有効な YAML が得られます。

9 次のコマンドを実行して、TKG クラスタをプロビジョニングします。

```
kubectl apply -f tkg2-cluster-1.yaml
```

予期される結果：

```
tanzukubernetescluster.run.tanzu.vmware.com/tkg2-cluster-1 created
```

10 TKG クラスタのプロビジョニングを監視します。

```
kubectl get tanzukubernetesclusters
```

```
kubectl get tkc
```

v1beta1 API を使用してクラスタを作成する場合は、次のコマンドを使用します。

```
kubectl get cluster
```

最初の READY ステータスは、クラスタがプロビジョニング中であるため False です。数分後、READY ステータスが True になります。

```
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
```

追加のコマンドを実行して、クラスタの詳細を表示します。

```
kubectl get
tanzukubernetescluster,cluster,virtualmachinesetresourcepolicy,virtualmachineservice,kubead
mcontrolplane,machinedeployment,machine,virtualmachine
```

```
kubectl describe tanzukubernetescluster tkg2-cluster-1
```

11 クラスタ ノードのデプロイを vSphere Client で監視します。

[ホストおよびクラスタ] に対応する vSphere インベントリに、ターゲット vSphere 名前空間 にデプロイされたクラスタ ノード仮想マシンが表示されます。

12 すべての TKG クラスタ ノードが準備完了状態になったら、kubectl 向けの vSphere プラグイン を使用してクラスタにログインします。

```
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
```

例：

```
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
```

注： ログイン コマンドは、制御プレーン ノードが実行され、認証サービス プラグインが起動した後にのみ成功します。ワーカー ノードの作成が進行中の場合にログインすると、結果にばらつきが生じる可能性があります。すべてのクラスタ ノードが準備完了状態になってから、ログインすることをお勧めします。

13 コンテキストを TKG クラスタに切り替え、これを現在のコンテキストにします。

TKG クラスタに正常にログインすると、次のようなメッセージが表示されます。

```
Logged in successfully.

You have access to the following contexts:
   192.197.2.65
   tkg2-cluster-namespace
   tkg2-cluster-1
```

ここで、`192.197.2.65` は スーパーバイザー コンテキスト、`tkg2-cluster-namespace` は vSphere 名前空間 コンテキスト、`tkg2-cluster-1` は TKG クラスタ コンテキストです。

TKG クラスタ コンテキストに切り替えます。

```
kubect config use-context tkg2-cluster-1
```

14 TKG クラスタ リソースを確認します。

```
kubectl get nodes
```

```
kubectl get namespaces
```

```
kubectl get pods -A
```

```
kubectl cluster-info
```

```
kubectl api-resources
```

15 テスト ポッドをデプロイして TKG クラスタを実行し、想定どおりに動作することを確認します。

Kubectl を使用した TKG クラスタ プロビジョニングのテストを参照してください。

Tanzu CLI を使用して TKG クラスタをプロビジョニングするためのワークフロー

このワークフローに従い、Tanzu CLI を使用して に v1beta1 TKG クラスタをプロビジョニングします。

前提条件

プロビジョニング ワークフローを開始する前に、次の前提条件を確認し、必要な手順を実施します。

- 使用環境に最新の スーパーバイザー バージョンをインストールするか、最新バージョンにアップデートします。2 章 TKG サービス クラスタの実行を参照してください。
- コンテンツ ライブラリを最新の Tanzu Kubernetes リリース で作成またはアップデートします。5 章 TKG サービス クラスタ用 Kubernetes リリースの管理を参照してください。
- TKG 2.0 クラスタをホストするための vSphere 名前空間 を作成して構成します。6 章 TKG サービス クラスタをホストするための vSphere 名前空間 の構成を参照してください。

デフォルトの TKG クラスタのプロビジョニング

Tanzu CLI を使用してデフォルトの v1beta1 クラスタをプロビジョニングするには、次の手順を実行します。その他のガイダンスやトラブルシューティングについては、スタンドアローン TKG のドキュメントの「ワークロード クラスタの作成」を参照してください。

1 Tanzu CLI をインストールします。

  TKG サービス クラスタで使用するための Tanzu CLI のインストールを参照してください。

2 Tanzu CLI を使用して スーパーバイザー に接続します。

  - Tanzu CLI と vCenter SSO 認証を使用した スーパーバイザー への接続を実行するか、または
  - Tanzu CLI と外部 ID プロバイダを使用した スーパーバイザー への接続

3 使用可能な TKr を一覧表示します。

```
tanzu kubernetes-release get
```

4 目的の構成を持つクラスタ マニフェストを作成します。

  vSphere 8 スーパーバイザー 上の TKG では、Tanzu CLI で Kubernetes 形式のオブジェクト仕様を使用して、ClusterClass に基づいてクラスタを作成できます。

  a まず、v1beta1 の例：デフォルト クラスタを開始します。

  b `spec.clusterNetwork` に必要な `cidrBlocks` をポピュレートします。

  c 表にリストされている予期される値を `spec.topology` にポピュレートします。

    - TKR NAME 文字列（`v1.26.13---vmware.1-fips.1-tkg.3` など）
    - 制御プレーン ノードの数（`3` など）
    - 各ワーカー ノード プールの名前（`node-pool-1` など）。

- ワーカー ノードの数（3 など）。
- 仮想マシン クラス（guaranteed-medium など）
- ストレージ クラス（tkg2-storage-policy など）

5 クラスタ マニフェストを cluster-default.yaml として保存し、YAML チェッカーを使用して検証します。

6 TKG クラスタを作成します。

```
tanzu cluster create -f cluster-default.yaml
```

注： 詳細を出力するには、-v 8 を追加します。

7 TKG クラスタが作成されていることを確認します。

```
Workload cluster 'cluster-default' created
```

8 クラスタが作成されたら、次のコマンドを実行してクラスタのステータスを確認します。

```
tanzu cluster get cluster-default
```

9 クラスタを一覧表示します。

```
tanzu cluster list
```

10 クラスタ ノードを確認します。

```
tanzu cluster machinehealthcheck node get cluster-default
```

```
tanzu cluster machinehealthcheck control-plane get cluster-default
```

11 TKG クラスタの構成コンテキストを取得します。

```
tanzu cluster kubeconfig get cluster-default -n tkg2-cluster-ns
```

12 クラスタにアクセスします。

```
kubectl config use-context tanzu-cli-cluster-default@cluster-default
```

13 テスト ポッドをデプロイして TKG 2.0 クラスタを実行し、想定どおりに動作することを確認します。

Kubectl を使用した TKG クラスタ プロビジョニングのテストを参照してください。

スーパーバイザー でのカスタム TKG クラスタのプロビジョニング

v1beta1 の例：Calico CNI を含むクラスタ などのカスタム v1beta1 クラスタをプロビジョニングするには、例に示されているように、すべての仕様を単一の YAML に配置し、使用環境に合わせて特定の値を変更し、kubectl apply -f cluster-calico.yaml などを実行します。

Tanzu CLI を使用して同じカスタム v1beta1 クラスタをプロビジョニングするには、クラスタを作成する前に構成オブジェクト CalicoConfig と ClusterBootstrap が存在している必要があります。

Calico CNI を使用してクラスタをプロビジョニングするには、次の手順を実行します。

1 `CalicoConfig` および `ClusterBootstrap` 構成オブジェクトの YAML を作成し、各 YAML で目的のクラスタ名と名前空間を指定します。

2 3 つの構成オブジェクトそれぞれに `kubectl apply -f` を実行するか、これらを 1 つの YAML に配置して `kubectl apply -f` を実行します。

3 構成オブジェクトと一致する名前および名前空間、およびその他の必要なパラメータが含まれているクラスタ仕様 `cluster-calico.yaml` を作成します。

4 クラスタを作成します。

```
tanzu cluster create -f cluster-calico.yaml
```

Kubectl を使用した TKG クラスタ プロビジョニングのテスト

TKG クラスタをプロビジョニングしたら、テスト ワークロードをデプロイして、クラスタの機能を確認することをお勧めします。

テスト アプリケーションをデプロイし、TKG クラスタが実行中であることを確認します。

前提条件

- TKG クラスタをプロビジョニングします。
- TKG クラスタに接続します。

手順

1 TKG クラスタをプロビジョニングします。

Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。

2 kubectl を使用して スーパーバイザー にログインします。

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

3 TKGS クラスタがプロビジョニングされている vSphere 名前空間 に構成コンテキストを切り替えます。

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

4 ターゲットの TKG クラスタにログインします。

```
kubectl vsphere login --server=<IP or FQDN> --vsphere-username <USERNAME> \
--tanzu-kubernetes-cluster-name CLUSTER-NAME \
--tanzu-kubernetes-cluster-namespace NAMESPACE-NAME
```

5 次の内容の `ping-pod.yaml` ファイルを作成します。

```
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 `regcred` レジストリ認証情報を作成します。

このシナリオで使用されるコンテナ イメージ (busybox) はパブリック Docker Hub レジストリからプルされますが、イメージのプルが制限されている場合があります。その場合は、Docker Hub アカウントと、ポッド仕様で参照されているイメージ プル シークレット（「regcred」）が必要になります。このシークレットを作成するには、「プライベート レジストリ認証情報シークレットの作成」を参照してください。

7 ポッド セキュリティを必要に応じて構成します。

Tanzu Kubernetes リリース v1.24 以前を使用している場合は、次の手順に進んでポッドを作成します。

Tanzu Kubernetes リリース v1.25 を使用している場合は、TKR 1.25 以降の PSA の構成が有効になります。次の手順に進んでポッドを作成できます。ただし、ポッドのセキュリティ違反に関する警告が表示されます。これは無視してかまいません。

```
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")
```

Tanzu Kubernetes リリース v1.26 以降を使用している場合は、TKR 1.25 以降の PSA の構成が適用されます。次の手順で示すようにポッドを作成すると失敗し、以下のエラーが表示されます。

```
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")
```

これを修正するには、ポッドが作成された `default` 名前空間で次のコマンドを実行します。これにより、名前空間の TKR 1.25 以降の PSA の構成が削除されます。

```
kubectl label --overwrite ns default pod-security.kubernetes.io/enforce=privileged
```

または、次のように securityContext を直接ポッドに適用することもできます。

```
...
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 YAML を適用します。

```
kubectl apply -f ping-pod.yaml
```

予期される結果：

```
pod/ping-pod created
```

9 ポッドが正常に完了したことを確認します。

```
kubectl get pods -n default
```

```
NAME       READY   STATUS      RESTARTS   AGE
ping-pod   0/1     Completed   0          13s
```

10 ポッドが DNS サーバに ping を送信したことを確認します。

```
kubectl logs ping-pod -f
```

予期される結果：

```
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 ポッドを削除します。

```
kubectl delete -f ping-pod.yaml
```

12 ポッドが削除されたことを確認します。

```
kubectl get pods
```

Kubectl または Tanzu CLI を使用した TKG クラスタの削除

TKG クラスタを削除するには、`kubectl` または Tanzu CLI を使用します。

`kubectl` または Tanzu CLI を使用して Tanzu Kubernetes クラスタを削除すると、Kubernetes ガベージ コレクションによってすべての依存リソースが削除されます。

注： vSphere Client または vCenter Server CLI を使用してクラスタを削除しないでください。

手順

1 スーパーバイザー で認証します。

2 削除する TKG がプロビジョニングされている vSphere 名前空間 にコンテキストを変更します。

```
kubectl config use-context CLUSTER-NAMESPACE
```

3 この名前空間内にある TKG クラスタを一覧表示します。

```
kubectl get clusters
```

```
tanzu cluster list
```

4 次の構文を使用して TKG クラスタを削除します。

kubectl を使用した v1alpha3 クラスタ：

```
kubectl delete tanzukubernetescluster --namespace CLUSTER-NAMESPACE CLUSTER-NAME
```

kubectl を使用した v1beta1 クラスタ：

```
kubectl delete cluster --namespace CLUSTER-NAMESPACE CLUSTER-NAME
```

Tanzu CLI を使用した v1alpha3 または v1beta1 クラスタ：

```
tanzu cluster delete --namespace CLUSTER-NAMESPACE CLUSTER-NAME
```

結果の例：

```
tanzukubernetescluster.run.tanzu.vmware.com "tkg-cluster-1" deleted
```

5 クラスタが削除されたことを確認します。

kubectl：

```
kubectl get clusters
```

Tanzu CLI：

```
tanzu cluster list
```

6 kubeconfig ファイルからクラスタのコンテキストを削除します。

```
kubectl config delete-context CONTEXT
```

例：

```
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
```

クラスタ v1beta1 API の使用

このセクションでは、v1beta1 API を使用してクラスタをプロビジョニングするためのリファレンス コンテンツを提供します。必要に応じてさまざまな構成やカスタマイズを行う例も含まれています。

クラスタ v1beta1 API

クラスタ v1beta1 API を使用すると、デフォルトの ClusterClass 定義に基づいてクラスタをプロビジョニングできます。

ClusterClass API v1beta1

Kubernetes クラスタ API は、Kubernetes クラスタの宣言型プロビジョニング、アップグレード、および運用を提供するツールのスイートです。ClusterClass は、一連のクラスタのライフサイクルを管理するためのテンプレートを定義できる、先進的なクラスタ API です。TKG サービス では、v1beta1 API の使用によって ClusterClass がサポートされます。

TKG サービス には、tanzukubernetescluster という名前のデフォルトの ClusterClass 定義が付属しています。tanzukubernetescluster ClusterClass では、v1beta API を使用してクラスタを作成するためのテンプレートが提供されます。tanzukubernetescluster ClusterClass は、すべてのユーザー名前空間で使用できます。この ClusterClass に基づいてクラスタを作成するには、クラスタ仕様で参照します。手順については、v1beta の例を参照してください。

デフォルトの ClusterClass「tanzukubernerescluster」

デフォルトの tanzukubernetescluster ClusterClass は変更できません。TKG サービスの各リリースで更新される可能性があります。

TKG サービス インスタンスが付属するデフォルトの tanzukubernetescluster ClusterClass を表示するには、次の手順を実行します。

1 スーパーバイザー にログインします。

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USER@vsphere.local
```

2 TKGS クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

```
kubeclt config use-context VSPEHRE-NS
```

3 デフォルトの tanzukubernetescluster ClusterClass を取得します。

```
kubectl get clusterclass tanzukubernetescluster -o yaml
```

4 必要に応じて、デフォルトの ClusterClass の出力を *tkc-dcc.yaml* という名前のファイルに書き込むことができます。

```
kubectl get clusterclass tanzukubernetescluster -o yaml > tkc-dcc.yaml
```

クラスタをカスタマイズするための ClusterClass 変数

変数を使用して、tanzukubernetescluster ClusterClass に基づいてクラスタをカスタマイズします。変数は、名前と値のペアを使用して定義されます。構文は、openAPIV3Schema に準拠している必要があります。

v1beta1 API を使用してクラスタをプロビジョニングするには、2 つの変数が必要です。

- 仮想マシン クラス
- ストレージ クラス

クラスタをカスタマイズするために、次のような変数がさらに使用可能です。

- プロキシ
- TLS 証明書
- SSH キー

以下セクションに、デフォルトの tanzukubernetescluster ClusterClass で使用できるすべての変数を示します。

重要： 有効なキー名は、英数字、ダッシュ（key-name など）、アンダースコア（KEY_NAME など）、ドット（key.name など）のみで構成する必要があります。キー名にスペースを使用することはできません。

clusterEncryptionConfigYaml

clusterEncryptionConfigYaml 変数は、クラスタの暗号化を構成する場合に使用します。

clusterEncryptionConfigYaml

暗号化構成の詳細を提供する YAML ファイルである文字列。

Kube-apiserver 暗号化構成パッケージを使用して、etcd データベースでデータの暗号化を構成できます。Kubernetes のドキュメントの Encrypting Secret Data at Rest を参照してください。

```
...
    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

controlPlaneCertificateRotation 変数は、制御プレーン ノードの TLS 証明書が期限切れになる前にそれらの証明書のロールアウトをトリガすることで、証明書をローテーションするようにシステムを構成する場合に使用します。新規と既存のすべての制御プレーン ノードで、制御プレーン証明書のローテーションを使用できます。

controlPlaneCertificateRotation

機能を有効にするかどうかを示すブール値および証明書のローテーションの期限切れまでの日数。詳細については、「Automatically rotating certificates using Kubeadm Control Plane provider」を参照してください。

```
...
    variables:
    - name: controlPlaneCertificateRotation
      value:
        activate: true
        daysBefore: 90
```

ここで、

- activate は、機能を有効にするかどうかを示すブール値です。デフォルトは true です。
- daysBefore は、期限切れまでの日数です。デフォルトは 90 日です。最小値は期限切れの 7 日前です。

注： この変数は、vSphere 8 Update 3 (TKG サービス 3.0) 用に更新されます。

controlPlaneVolumes

controlPlaneVolumes 変数は、制御プレーン ノードのパーシステント ボリュームを構成する場合に使用します。

controlPlaneVolumes

オブジェクトのオプションの配列。それぞれに name、storageClass、および mountPath が含まれ、それぞれが文字列であり、およびオプションの storage 文字列が含まれる capacity オブジェクトです。

```
...
    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

defaultRegistrySecret 変数は、クラスタのデフォルトのコンテナ レジストリを構成します。

注： この変数は、組み込みの Harbor レジストリで使用するために予約されています。

defaultRegistrySecret

デフォルトのコンテナ レジストリのパブリック キー、証明書名、および名前空間を含むオブジェクト。

スーパーバイザー で組み込みの Harbor レジストリを有効にすると、defaultRegistrySecret 変数はクラスタが信頼するレジストリ サービス証明書を指定します。証明書シークレットには、managed-by: vmware-vRegistry というラベルが付いています。クラスタの作成時に、defaultRegistry 証明書が defaultRegistrySecret 変数に挿入されます。クラスタの作成後、変数を手動で更新して証明書のローテーションまたは更新を管理します。

```
...
    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

`defaultStorageClass` 変数は、クラスタのデフォルト ストレージ クラスを構成する場合に使用します。

defaultStorageClass

デフォルトのストレージ クラスとして使用するストレージ クラスを特定する文字列。Helm チャートや Tanzu パッケージなどの特定のアプリケーションで必要になることがあります。

```
...
    variables:
    - name: defaultStorageClass
      value: tkg2-storage-profile
```

extensionCert

`extensionCert` 変数は、TLS 証明書を構成する場合に使用します。

extensionCert

`name` 文字列と `key` 文字列を含む `contentSecret` オブジェクトを含むオブジェクト。`contentSecret` は、TLS 証明書用に作成された Kubernetes シークレット オブジェクトを参照します。

```
...
    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

`kubeAPIServerFQDNs` 変数は、FQDN を使用してクラスタを構成する場合に使用します。

kubeAPIServerFQDNs

1 つ以上の完全修飾ドメイン名 (FQDN) の配列。

生成される Kubernetes API 証明書には、`kubeAPIServerFQDNs` 変数で指定した各 FQDN が含まれます。システムは kubeconfig にリストの最初の FQDN をポピュレートし、それが解決可能であるとみなします。リストの別の FQDN を使用する場合は、変数リストの目的の FQDN を使用して、生成された kubeconfig ファイルを手動で編集できます。

詳細については、v1beta1 の例：FQDN を使用するクラスタを参照してください。

nodePoolLabels

`nodePoolLabels` 変数は、ワーカー ノードのラベルを構成する場合に使用します。

nodePoolLabels

1 つ以上のオブジェクトの配列。各オブジェクトにはキーと値のペア（両方とも文字列）が含まれています。

ラベルを使用すると、クエリやレポートを実行しやすくするために、要件に応じてシステム オブジェクトを管理できます。使用方法の詳細については、Kubernetes ラベルのドキュメントを参照してください。

nodePoolTaints

`nodePoolTaints` 変数は、テイントをワーカー ノードに適用する場合に使用します。

nodePoolTaints

オブジェクトの配列。各オブジェクトには、ワーカーノードに適用されるテイントが含まれています。

各テイント オブジェクトには、`key`（文字列）、`value`（文字列）、および `effect`（文字列）が含まれます。作成または更新時に、`timeAdded` フィールドにポピュレートされます。

nodePoolVolumes

`nodePoolVolumes` 変数は、クラスタ ノードのパーシステント ボリュームを指定する場合に使用します。

nodePoolVolumes

オブジェクトのオプションの配列。それぞれに `name`、`storageClass`、および `mountPath` が含まれ、それぞれが文字列であり、オプションの `storage` 文字列が含まれる `capacity` オブジェクトです。

```
...
    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

`ntp` 変数は、クラスタの NTP サーバを構成する場合に使用します。

ntp

NTP サーバの FQDN または IP アドレスを表す文字列。

NTP 変数は、例にあるように、NTP サーバのドメイン名を指定します。NTP サーバは、クラスタの作成時にクラスタ変数に挿入されます。クラスタの作成後、クラスタ変数を手動で更新して、サーバ名のローテーションまたは更新を管理します。

```
...
    variables:
    - name: ntp
      value: time1.vmware.com
```

podSecurityStandard

`podSecurityStandard` 変数は、クラスタ全体のポッド セキュリティを構成する場合に使用します。

注： これは、vSphere 8 Update 3 (TKG サービス 3.0) の新しい変数です。

podSecurityStandard

TKr v1.26 以降では、注釈ラベルを使用して名前空間レベルでポッド セキュリティ (PSA) の制限がデフォルトで適用されます。TKR 1.25 以降の PSA の構成を参照してください。

`podSecurityStandard` 変数を使用して、v1beta1 クラスタをプロビジョニングまたは更新するときにクラスタ全体の PSA を構成することもできます。

`podSecurityStandard` 変数は次のように実装できます。

```
...
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]
```

ここで、

- DEACTIVATED 値は、クラスタ全体の PSA を適用する場合は `false`（デフォルト）、それ以外の場合は `true` です。
- *-PROFILE 値は、各モードの PSA プロファイルです。`"privileged"`、`"baseline"`、または `"restricted"`（デフォルト）を指定できます。
- *-VERSION 値は、各モードの Kubernetes バージョン（例：`"v1.26"`）です。値 `"latest"` がデフォルトです。
- EXEMPT-NS 値は、PSA 制御から除外する名前空間のカンマ区切りのリストです。

注： システム名前空間は、kube-system、tkg-system、vmware-system-cloud-provider などのポッド セキュリティから除外されます。

`podSecurityStandard` 変数を実装しない場合は、デフォルトの PSA の動作が保持されます。クラスタ仕様に `podSecurityStandard` 変数を含めると、変数の設定が（オーバーライドしない限りデフォルトも含めて）制御されます。

次の例では、デフォルトが表示されます。

```
...
    variables:
      - name: podSecurityStandard
        value:
          enforce: "restricted"
          enforce-version: "latest"
```

次の例では、現在のポッド セキュリティ強化のベスト プラクティスに従っていないワークロードを特定するための監査ログと警告を提供しますが、既知の権限の昇格を許可しない最小限の制限があるポリシー（「ベースライン」）のみを適用します。

```
...
    variables:
      - name: podSecurityStandard
        value:
          audit: "restricted"
          warn: "restricted"
          enforce: "baseline"
```

次の例では、特定の名前空間を除き、制限付きポリシーを適用します。

```
...
    variables:
      - name: podSecurityStandard
        value:
          audit: "restricted"
          warn: "restricted"
          enforce: "restricted"
          exemptions:
            namesaces: ["privileged-workload-ns"]
```

次の例では、特定の TKr バージョンへの適用を制限します。

```
...
variables:
  - name: podSecurityStandard
    value:
      audit-version: "v1.26"
      warn-version: "v1.26"
      enforce-version: "v1.26"
```

その他の例については、Kubernetes ドキュメントのポッド セキュリティ標準を参照してください。

proxy

`proxy` 変数は、クラスタのプロキシ サーバを構成する場合に使用します。

proxy

送信クラスタ接続のプロキシ サーバを参照するパラメータを持つオブジェクト。

必要な `proxy` パラメータは `httpProxy`、`httpsProxy`、および `noProxy` です。クラスタ定義に `proxy` 変数を含める場合、3 つのフィールドはすべて必須です。

`httpProxy` フィールドと `httpsProxy` フィールドでは、TKG クラスタからの送信 HTTP および HTTPS 接続を管理するように構成されているプロキシ サーバの URI を参照する文字列値が使用されます。HTTP を使用してプロキシ サーバに接続できます。HTTPS 接続はサポートされていません。

`noProxy` フィールドは文字列の配列です。`noProxy` 値を スーパーバイザー ワークロード ネットワークから取得します。名前空間ネットワーク、Ingress、および Egress サブネットを `noProxy` フィールドに含める必要があります。

`noProxy` フィールドにサービス サブネットを含める必要はありません。TKG クラスタは、このサブネットと連携しません。

`noProxy` フィールドに `clusterNetwork.services.cidrBlocks` および `clusterNetwork.pods.cidrBlocks` を含める必要はありません。これらのエンドポイントは自動的にプロキシされません。

`noProxy` フィールドに localhost および `127.0.0.1` を含める必要はありません。これらのエンドポイントは自動的にプロキシされません。

```
...
    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

`storageClass` 変数は、クラスタのストレージ クラスを構成する場合に使用します。

storageClass

TKG クラスタがプロビジョニングされている vSphere 名前空間 に割り当てられている vSphere ストレージ プロファイルの名前である文字列。

```
...
    variables:
    - name: storageClass
      value: tkgs-storage-profile
```

次のコマンドを使用して、使用可能なストレージ クラスを一覧表示します。

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

または、vSphere 管理者権限がある場合は、次の手順を実行します。

```
kubectl describe storageclasses
```

storageClasses

`storageClasses` 変数は、クラスタのストレージ クラスの配列を構成する場合に使用します。

storageClasses

1 つ以上の文字列の配列。各文字列は、TKG クラスタがプロビジョニングされている vSphere 名前空間 に割り当てられている vSphere ストレージ プロファイルの名前です。

```
...
    variables:
    - name: storageClasses
      value: [tkg2-storage-profile, tkg2-storage-profile-latebinding]
```

次のコマンドを使用して、使用可能なストレージ クラスを一覧表示します。

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

または、vSphere 管理者権限がある場合は、次の手順を実行します。

```
kubectl describe storageclasses
```

TKR_DATA

`TKR_DATA` 変数は、TKR 情報を指定する場合に使用します。

TKR_DATA

TKR バージョンおよびその他詳細の指定に使用するオブジェクト。

`version` は、クラスタ ノードで使用される TKR NAME 形式の文字列です。

`legacy-tkr` ラベルを持たない TKR のみが vSphere 9 スーパーバイザー上の TKG と互換性があります。TKG サービス クラスタでの Kubernetes リリースの使用を参照してください。

デフォルトのオペレーティング システムは PhotonOS です。注釈を使用して、v1beta1 の例：Ubuntu TKR を使用するクラスタを指定します。

trust

`trust` 変数は、クラスタの信頼できる CA 証明書を 1 つ以上指定する場合に使用します。

trust

クラスタに TLS 証明書（追加の CA またはエンド証明書のいずれか）を追加するためのオブジェクト。

値は、文字列の配列を保持する `additionalTrustedCAs` です。例：

```
#trust-example
    variables:
      - name: trust
        value:
          additionalTrustedCAs:
          - name: additional-ca-1
          - name: additional-ca-2
          - name: additional-ca-N
```

各文字列の値は、double 型の base64 エンコードの PEM 形式の CA 証明書を含む Kubernetes シークレットのデータ マップ フィールドのユーザー定義名です。例：

```
#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
```

trust 変数の一般的な使用事例は、v1beta1 クラスタとプライベート コンテナ レジストリを統合することです。TKG サービス クラスタとプライベート コンテナ レジストリの統合を参照してください。

user

`user` 変数は、クラスタのユーザー認証情報を指定する場合に使用します。

user

名前文字列とキー文字列、および sshAuthorizedKey 文字列を持つ、passwordSecret オブジェクトを含むオブジェクト。この変数を使用して、ユーザーの SSH キーをリモート SSH アクセス用のクラスタ ノードに追加できます。

ユーザー変数は、パスワードと認証済みキーを含む SSH ログイン認証情報を指定します。デフォルトのユーザー名は `vmware-system-user` です。パスワードはハッシュされ、クラスタがプロビジョニングされているのと同じ名前空間のシークレットに格納される必要があります。`passwordSecret` オブジェクトはこのシークレットを参照します。たとえば、Linux では、`mkpasswd --method=SHA-512 --rounds=4096` を使用して安全なハッシュを生成できます。詳細については、Including users and groups を参照してください。

```
...
    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

`vmClass` 変数は、クラスタ ノードの仮想マシン クラスを構成する場合に使用します。

vmClass

TKG クラスタがプロビジョニングされている vSphere 名前空間 にバインドされている仮想マシン クラスの名前にマッピングされる必須の文字列。

`vmClass` は、クラスタ ノードに使用される仮想ハードウェア設定を記述する VirtualMachineClass の名前です。VirtualMachineClass は、ノードで使用可能な CPU とメモリ、およびこれらのリソースの要求と制限を制御します。TKG サービス クラスタでの仮想マシン クラスの使用を参照してください。

TKG サービス クラスタがプロビジョニングされている vSphere 名前空間 に関連付けられている仮想マシン クラスのみを使用できます。バインドされたクラスを一覧表示するには、`kubectl get virtualmachineclass` コマンドを使用します。

制御プレーン ノードとノード プール ワーカー ノードに異なる仮想マシン クラスを使用できるよう、さまざまな範囲で `vmClass` 変数を定義できます。

たとえば、ここではインラインの `vmClass` 変数は、この特定の machineDeployment トポロジに対するプライマリの `vmClass` 変数を `overrides`（オーバーライド）します。

```
...
    workers:
      machineDeployments:
      - class: tkg-worker
        name: compute
        replicas: 3
        variables:
          overrides:
```

```
          - name: vmClass
            value: guaranteed-large
```

v1beta1 の例：デフォルト クラスタ

デフォルト設定を使用して v1beta1 クラスタをプロビジョニングするには、この例を参照してください。

v1beta1 の例：デフォルト クラスタ

次のサンプル YAML では、v1beta1 API を使用して、デフォルトの ClusterClass に基づいてデフォルトのクラスタを作成します。

この例は、v1beta1 API を使用してクラスタを作成するために必要な最小の構成を表しています。例には、各フィールドの説明が注釈として付いています。詳細については、ソース コードを参照してください。

この例に関する次の点にご注意ください。

- TanzuKubernetesCluster v1alpha3 API と異なり、v1beta1 API では `clusterNetwork` を指定する必要があります。クラスタ タイプにデフォルトのネットワーク設定はありません。
- デフォルトの ClusterClass は、「クラスタ v1beta1 API 」に記載されている `tanzukubernetescluster` です。
- クラスタをカスタマイズするには、それぞれが名前と値のペアである `variables` を使用します。例に示すように、少なくとも仮想マシンとストレージ クラスを変数として指定する必要があります。
- Tanzu パッケージや Helm のチャートなどの多くのワークロードではデフォルトのストレージ クラスを使用してクラスタをプロビジョニングする必要があるため、この例には `defaultStorageClass` 変数も含まれています（ただし、技術的には省略可能です）。

```
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 の例：デフォルトの ClusterClass に基づくカスタム クラスタ

カスタム設定を使用して v1beta1 クラスタをプロビジョニングするには、この例を参照してください。

v1beta1 の例：デフォルトの ClusterClass に基づくカスタム クラスタ

次のサンプル YAML では、v1beta1 API を使用して、変数を使うカスタム設定をいくつか使用してクラスタをプロビジョニングする方法を示します。この例は、v1beta1 の例：デフォルト クラスタを基準にしています。

この例では、containerd や kubelet などの変更の多いコンポーネントについて、ワーカー ノード上のパーシステント ボリュームに対する値を使用しています。また、`vmClass` 変数は 2 回宣言されます。`vmClass` で宣言された `workers.machineDeployments` 変数が、グローバルに宣言された `vmClass` 変数を上書きし、ワーカー ノードがより大きな仮想マシン クラスを使用してプロビジョニングされるようにします。

```
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 の例：Calico CNI を含むクラスタ

デフォルトの Antrea CNI ではなく Calico CNI を使用して v1beta1 クラスタをプロビジョニングするには、この例を参照してください。クラスタの 1 つ以上の TKR パッケージをカスタマイズする際のガイダンスについても、この例を参照してください。

v1beta1 の例：カスタム CNI を含むクラスタ

次のサンプル YAML では、v1beta1 API を使用してカスタム CNI を含むクラスタをプロビジョニングする方法を示します。この例は、v1beta1 の例：デフォルト クラスタを基準にしています。

tanzukubernetescluster ClusterClass で定義されているように、デフォルトの CNI は Antrea です。サポートされているその他の CNI は Calico です。CNI を Antrea から Calico に変更するには、CalicoConfig カスタム リソースを参照する ClusterBootstrap カスタム リソースを作成して、デフォルトの CNI に負荷をかける必要があります。

ClusterBootstrap カスタム リソースには、TKR から取得された値を持つ spec.cni.refName ブロックが含まれています。(このフィールドの値を取得する方法のガイダンスについては、TKr パッケージを参照してください。) ClusterBootstrap 値は ClusterClass のデフォルト値を上書きし、クラスタの作成時に Cluster API (CAPI) によって選択されます。ClusterBootstrap カスタム リソースの名前は、Cluster と同じである必要があります。

注： この例は単一の YAML ファイルとして提供されますが、個別のファイルに分割することができます。分割する場合は、最初に CalicoConfig カスタム リソース、次に ClusterBootstrap、cluster-calico クラスタの順に作成する必要があります。

```
---
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 の例：Ubuntu TKR を使用するクラスタ

Tanzu Kubernetes リリース の Ubuntu エディションを使用する v1beta1 クラスタをプロビジョニングするには、この例を参照してください。

v1beta1 の例：Ubuntu TKR を使用するクラスタ

次のサンプル YAML では、v1beta1 API を使用して、指定された TKR の Ubuntu エディションを使用するクラスタをプロビジョニングする方法を示します。この例は、v1beta1 の例：デフォルト クラスタを基準にしています。

デフォルトでは、クラスタ ノードには PhotonOS が使用されます。TKR バージョンで複数の OSImage がサポートされている場合に、Photon ではなく Ubuntu を使用するには、クラスタ仕様に注釈 `run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu` を含めます。TKr の Ubuntu エディションを使用している場合は、フルバージョンの文字列を指定し、クラスタ仕様に OS の注釈を含める必要があります。TKR の詳細については、5 章 TKG サービス クラスタ用 Kubernetes リリースの管理を参照してください。

```
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 の例：FQDN を使用するクラスタ

1 つ以上の完全修飾ドメイン名 (FQDN) を使用して TKGS クラスタをプロビジョニングするには、この例を参照してください。

FQDN のサポート

v1beta1 API では、完全修飾ドメイン名 (FQDN) を使用して TKG クラスタをプロビジョニングできます。クラスタ v1beta1 API には、Kubernetes API サーバ用の TLS 証明書で生成される 1 つ以上の FQDN 文字列を含む、`kubeAPIServerFQDNs` という名前の変数が含まれています。

FQDN が構成されているクラスタに対して `kubectl vsphere login` コマンドを実行すると、認証サービスではリスト内の最初の FQDN エントリを選択し、クラスタ操作のために優先する選択肢として kubeconfig に追加します。リスト内の最初の FQDN が解決可能であることが前提となります。クラスタのログインに必要な変更はありません。

生成される Kubernetes API 証明書には、`kubeAPIServerFQDNs` 変数で指定したすべての FQDN が含まれます。システムでは、リスト内の最初の FQDN 以外の FQDN の使用を試行しません。また、FQDN の解決も試行しません。リストの別の FQDN を使用する場合は、生成された kubeconfig ファイルを手動で編集して目的の FQDN を追加できます。

FQDN の要件

FQDN の使用はオプションです。FQDN を使用しない場合、機能は変更されません。ここで説明する機能は、TKG ワークロード クラスタに固有のものです。スーパーバイザー で FQDN を使用するには、スーパーバイザー のドキュメントで該当するトピックを参照してください。

FQDN を使用して TKG クラスタをプロビジョニングするには、次の要件を満たす必要があります。

- vSphere 8.0 U2 P03 以降の環境

- スーパーバイザー が最新のパッチ リリースに更新されている
- v1beta1 API クラスタのみがサポートされている（v1alpha3 API クラスタがサポートされていない）
- 選択した FQDN を有効な IP アドレスに解決するように DNS が構成されている

重要： FQDN 機能を使用できるのは、v1beta1 API を使用して CAPI クラスタをプロビジョニングしている場合のみです。FQDN で v1alpha3 API を使用して TKC をプロビジョニングすることはできません。

FQDN の例

クラスタ v1beta1 API を使用して、FQDN を持つクラスタを作成します。

`spec.topology.variables.kupeAPIServerFQDNs` 値は FQDN の配列です。

リスト内の最初の FQDN（この例では `demo.fqdn.com`）がシステムによって選択されます。

```
#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 の確認

変数リスト内の最初の FQDN が `kubeconfig` ファイルに含まれていること、および変数リスト内のすべての FQDN が Kubernetes API サーバ用の TLS 証明書に含まれていることを確認する場合は、次の手順を実行します。

1 Kubectl を使用して TKG クラスタにログインします。

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

2 `kubeconfig` ファイルの FQDN を表示します。

```
cat ~/.kube/config
```

3 リスト内の最初の FQDN 変数が `kubeconfig` に含まれていることを確認します。

例：

```
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 vSphere Client を使用して、[ワークロード管理] - [*名前空間*] - [コンピューティング] - [Tanzu Kubernetes クラスタ] - [制御プレーンのアドレス] で TKGS クラスタの IP アドレスを取得します。

5 IP アドレスと FQDN を使用して、ローカルの `/etc/hosts` ファイルに手動で DNS エントリを作成します。

例：

```
sudo vi /etc/hosts
127.0.0.1 localhost
127.0.1.1 ubuntu-client-vm
10.199.155.77 demo.fqdn.com
...
```

6 `openssl s_client` コマンドを使用して TLS 証明書を表示します。

```
echo | openssl s_client -servername hostname -connect FQDN:PORT 2>/dev/null | openssl x509
-text
```

`FQDN` は、`kubeAPIServerFQDNs` 変数リスト内の最初の FQDN です。

例：

```
echo | openssl s_client -servername hostname -connect demo.fqdn.com:6443 2>/dev/null |
openssl x509 -text
```

7 `Subject Alternative Name` フィールドには、すべての FQDN が含まれています。

```
X509v3 Subject Alternative Name:
    DNS:demo.fqdn.com, DNS:explore.fqdn.com, DNS:kubernetes, DNS:kubernetes.default,
DNS:kubernetes.default.svc, DNS:kubernetes.def
```

Kubernetes API サーバ用の TLS 証明書には `kubeAPIServerFQDNs` リスト内のすべての FQDN が含まれているため、リスト内の 2 番目（または 3 番目。以下も同様）の FQDN を使用するように `kubeconfig` ファイルを手動で更新できます。これにより、指定した FQDN が使用されます（解決可能であることが前提）。

v1beta1 の例：複数の vSphere Zone を横断するクラスタ

3 つの スーパーバイザー にデプロイされた vSphere Zones に v1beta1 クラスタをプロビジョニングするには、この例を参照してください。

v1beta1 の例：複数の vSphere Zone を横断するクラスタ

次のサンプル YAML では、v1beta1 API を使用して、vSphere Zone トポロジ内のクラスタをプロビジョニングします。この例は、v1beta1 の例：デフォルト クラスタを基準にしています。

この例では、複数のワーカー ノード プールを実装します。各ノード プールは、vSphere Zone にマッピングされている障害ドメインを表しています。vSphere Zones の詳細については、vSphere IaaS 制御プレーンのインストールと構成を参照してください。

```
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 の例：ルーティング可能なポッド ネットワークを使用するクラスタ

v1beta1 API を使用して、ルーティング可能なポッド ネットワークが構成されたクラスタを作成できます。そのためには、デフォルト クラスタを `AntreaConfig` および `VSphereCPIConfig` のカスタム構成でオーバーライドします。

v1beta1 API を使用したルーティング可能なポッド ネットワークについて

次のサンプル YAML では、v1beta1 API を使用して Antrea のルーティング可能なポッドが有効なクラスタをプロビジョニングする方法を示します。この例は、「v1beta1 の例：デフォルト クラスタ」を基準にしています。

ルーティング可能なポッド機能を有効にするには、特別な構成の `AntreaConfig` と `VSphereCPIConfig` がクラスタに必要です。

`AntreaConfig` では、`trafficEncapMode: noEncap` と `noSNAT: true` を設定する必要があります。

`VSphereCPIConfig` では、`antreaNSXPodRoutingEnabled: true`、`mode: vsphereParavirtualCPI`、および以下の設定が必要です。

```
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
```

`AntreaConfig` の名前の形式は、`<cluster-name>-antrea-package` にする必要があります。
`VSphereCPIConfig` の名前の形式は、`<cluster-name>-vsphere-cpi-package` にする必要があります。

構成ファイルが作成されたら、構成ファイルを参照するクラスタ仕様オブジェクトを作成します。クラスタの作成時に、構成ファイルを使用してクラスタをプロビジョニングし、デフォルトの構成を上書きします。

ルーティング可能なポッド ネットワークの作成：スーパーバイザー 構成

ルーティング可能なポッド ネットワークを作成するには、スーパーバイザー と TKG クラスタで構成を行う必要があります。

注： ルーティング可能なポッド ネットワークを使用するには、NSX に スーパーバイザー を構成する必要があります。VDS ネットワークでは、ルーティング可能なポッドを使用することはできません。

ルーティング可能なポッド ネットワークを スーパーバイザー で構成するには、次の手順を実行します。

1 新しい vSphere 名前空間 を作成します。

TKG サービス クラスタをホストするための vSphere 名前空間 の作成を参照してください。

2 [スーパーバイザー ネットワーク設定のオーバーライド] チェックボックスをオンにします。

詳細については、vSphere 名前空間のワークロード ネットワーク設定のオーバーライドを参照してください。

3 ルーティング可能なポッド ネットワークを次のように構成します。

フィールド	説明
[NAT モード]	ネットワーク アドレス変換 (NAT) を無効にするには、このオプションを選択解除します。
[名前空間ネットワーク]	このフィールドには、ルーティング可能な IP サブネットが IP アドレス/ビット（例：10.0.0.6/16）の形式でポピュレートされます。 NCP によって、ネットワークに指定された IP ブロックから 1 つ以上の IP プールが作成されます。 少なくともサブネット サイズ /23 を指定する必要があります。たとえば、サブネット プリフィックスが /28 のルーティング可能なサブネット /23 を指定すると、6 ノード クラスタに十分な数である 32 個のサブネットが取得されます。プリフィックスが /28 の /24 サブネットを指定した場合に取得されるサブネットは 2 個であり、これでは不十分です。 **注目：** ルーティング可能な IP サブネットを追加した後で、それがクラスタ ノードの IP アドレスを割り当てる [サービス CIDR] と重複していないことを確認します。[サービス CIDR] は、[スーパーバイザー] - [構成] - [ネットワーク] - [ワークロード ネットワーク] で確認できます。
[名前空間サブネット プリフィックス]	たとえば、/28 の形式でサブネット プリフィックスを指定します。 サブネット プリフィックスは、名前空間ネットワークから各ノードのポッド サブネットを取得するために使用されます。

4 [作成] をクリックして、ルーティング可能なポッド ネットワークを作成します。

ルーティング可能なポッド ネットワークの作成：TKG クラスタの構成

次のサンプル YAML は、ルーティング可能なポッド ネットワークを使用して v1beta1 クラスタを構成する方法を示します。

以下の例に示すように、ポッドの IP アドレスは cloud-provider-vsphere によって割り当てられるため、クラスタ仕様から `spec.clusterNetwork.pod` セクションを削除する必要があります。

注： この例は単一の YAML ファイルとして提供されますが、個別のファイルに分割することができます。分割する場合は、最初に `AntreaConfig` および `VSphereCPIConfig` カスタム リソース、次に target-cluster クラスタの順に作成する必要があります。

```
---
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 の例：SSL/TLS 用の追加の信頼できる CA 証明書を含むクラスタ

追加の信頼できる CA 証明書を 1 つ以上含む v1beta1 クラスタをプロビジョニングするには、この例を参照してください。

v1beta1 の例：追加の信頼できる CA 証明書を含むクラスタ

クラスタ v1beta1 API は、追加の信頼できる CA 証明書を 1 つ以上含むクラスタをプロビジョニングするための trust 変数を提供します。

表 7-1. v1beta1 API の trust 変数

フィールド	説明
`trust`	セクション マーカー。データを受け入れない。
`additionalTrustedCAs`	セクション マーカー。それぞれの `name` を示す証明書の配列が含まれます。
`name`	double 型の base64 エンコードの PEM 形式の CA 証明書を含む Kubernetes シークレットの `data` マップ フィールドのユーザー定義名。 **注：** base64 二重エンコードは必須です。内容が double 型の base64 エンコード形式でない場合は、結果の PEM ファイルを処理できません。

次の例は、CA 証明書を含む Kubernetes シークレットを v1beta1 API クラスタ仕様に追加する方法を示しています。

```
#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
```

次の例は、追加の信頼できる CA 証明書を含む Kubernetes シークレットを示しています。

```
#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
```

ここで、

- シークレットの `data` マップの値は、ユーザー定義の文字列です。これは CA 証明書の名前（この例では `additional-ca-1`）であり、値は double 型の base64 エンコードの PEM 形式の CA 証明書です。
- `metadata` セクションでは、シークレットの名前を *`CLUSTER-NAME`*`-user-trusted-ca-secret` にする必要があります。*CLUSTER-NAME* はクラスタの名前です。このシークレットは、クラスタと同じ vSphere 名前空間 で作成する必要があります。

CA 証明書の内容を double 型の base64 エンコード形式にします。

- Linux：`base64 -w 0 ca.crt | base64 -w 0`
- Windows：https://www.base64encode.org/

手順：新しいクラスタ

新しい TKGS クラスタに追加の信頼できる CA 証明書を 1 つ以上含める場合は、次の手順を実行します。

1 CA 証明書の内容を double 型の base64 エンコード形式にします。

2 データ マップ名を含む Kubernetes シークレットを作成します。値は double 型の base64 エンコードの PEM 形式の CA 証明書です。

3 クラスタ仕様で、`trust.additionalTrustedCAs` 変数にデータ マップの名前をポピュレートします。

4 通常どおりクラスタをプロビジョニングします。

Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。

5 クラスタが正常にプロビジョニングされると、追加した CA 証明書がクラスタによって信頼されます。

手順：既存のクラスタ

既存のクラスタに追加の信頼できる CA 証明書を 1 つ以上追加する場合は、次の手順を実行します。

1 CA 証明書の内容を double 型の base64 エンコード形式にします。

2 データ マップ名を含む Kubernetes シークレットを作成します。値は double 型の base64 エンコードの PEM 形式の CA 証明書です。

3 kubectl の編集が構成されていることを確認します。

kubectl のテキスト エディタの構成を参照してください。

4 クラスタ仕様を編集します。

```
kubectl edit tanzukubernetescluster/tkgs-cluster-name
```

5 `trust.additionalTrustedCAs` セクションを仕様に追加します。

6 double 型の base64 エンコードの PEM 形式の CA 証明書を含むシークレット内のデータ マップの名前を `additionalTrustedCAs` フィールドにポピュレートします。

7 テキスト エディタで変更を保存し、変更が kubectl によって登録されたことを確認します。

```
kubectl edit cluster/tkgs-cluster-name
cluster.run.tanzu.vmware.com/tkgs-cluster-name edited
```

8 クラスタのローリング アップデートが開始されると、信頼できる CA 証明書が追加されます。

TKG サービス クラスタのローリング アップデート モデルについてを参照してください。

追加の信頼できる CA 証明書の確認

クラスタに追加された信頼できる CA 証明書は、クラスタの kubeconfig ファイルに含まれています。

証明書のローテーション

証明書をローテーションするには、新しいシークレットを作成し、適切な値を使用してクラスタ仕様を編集します。これにより、クラスタのローリング アップデートがトリガされます。

注： システムは、*CLUSTER-NAME*-user-trusted-ca-secret への変更を監視しません。data マップ値が変更された場合は、クラスタに反映されません。新しいシークレットを作成して、そのデータ マップの name を trust.additionalTrustCAs にする必要があります。

追加の信頼できる CA 証明書のトラブルシューティング

追加の信頼できる CA に関するエラーのトラブルシューティングを参照してください。

使用事例

最も一般的な使用事例は、コンテナ レジストリに接続するための信頼できる CA を追加する場合です。TKG サービス クラスタとプライベート コンテナ レジストリの統合を参照してください。

v1beta1 の例：カスタム ClusterClass に基づくクラスタ（vSphere 8 U2 以降のワークフロー）

カスタム ClusterClass に基づいて TKG クラスタをプロビジョニングするには、次の手順を参照してください。これらの手順は、vSphere 8 U2 以降の環境に固有であることに注意してください。

前提条件

カスタム ClusterClass に基づいて TKG クラスタをプロビジョニングする手順は、vSphere 8 U2 リリースで更新されています。

次の前提条件を満たす必要があります。

- vSphere 8 U2 以降の環境
- ワークロード管理が有効
- スーパーバイザー 構成済み
- vSphere 向け Kubernetes CLI Tools がインストールされている Ubuntu クライアント

注目： カスタム ClusterClass は、アップストリーム クラスタ API のドキュメントに基づく Kubernetes の試験的な機能です。カスタム ClusterClass で使用可能なカスタマイズの範囲により、VMware は可能なすべてのカスタマイズをテストまたは検証できません。ユーザーは、カスタム ClusterClass クラスタのテスト、検証、およびトラブルシューティングを行う必要があります。カスタム ClusterClass クラスタに関するサポート チケットを発行できますが、VMware サポートはベスト エフォートベースにのみ制限され、カスタム ClusterClass クラスタに対して発行されたすべての問題の解決を保証することはできません。本番環境にカスタム ClusterClass クラスタを展開する前に、これらのリスクを認識しておく必要があります。

ワークフローの概要

ワークフローの概要は次のとおりです。

次のワークフローは、開始する必要があるすべてです。

手順	タスク	方法
1	デフォルトの ClusterClass のクローンを作成して、カスタム ClusterClass を作成します。	1: カスタム ClusterClass の作成
2	カスタム ClusterClass に基づいて新しい TKG クラスタをプロビジョニングし、すべてのクラスタ ノードが適切に起動していることを確認します。	2: カスタム ClusterClass に基づく TKG クラスタの作成

カスタム ClusterClass に変更を加え、カスタム ClusterClass クラスタ ノードのローリング アップデートを開始するには、次のワークフローを参照してください。

注： 次のワークフローで示す操作は、カスタム ClusterClass に対して実行できる操作の例です。ユースケースは異なる場合がありますが、一般的なワークフローを適用する必要があります。

手順	タスク	方法
3	ワーカー ノードの 1 つに SSH 接続して、更新するパッケージがあることを確認します。	3: パッケージ更新の有無の確認
4	更新を実行する新しいコマンドを使用して、カスタム ClusterClass を更新します。	4: カスタム ClusterClass の更新
5	更新がすでに実行されている新しいノードのロールアウトを確認します。	5: クラスタ ノードのローリング アップデートの確認

1: カスタム ClusterClass の作成

最初のパートでは、`tanzukubernetescluster` という名前のデフォルトの ClusterClass のクローンを作成することで、`ccc`（`customclusterclass` の省略形）という名前のカスタム ClusterClass を作成します。

注： カスタム ClusterClass 名はユーザー定義です。別の名前を使用する場合は、指示を適宜調整します。

1 *ccc-ns* という名前の vSphere 名前空間 を作成します。

権限、ストレージ、コンテンツ ライブラリ、および仮想マシン クラスを構成します。必要に応じて、6 章 TKG サービス クラスタをホストするための vSphere 名前空間 の構成を参照してください。

注： vSphere 名前空間 名はユーザー定義です。別の名前を使用する場合は、指示を適宜調整します。

2 スーパーバイザー にログインします。

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USER@vsphere.local
```

3 デフォルトの ClusterClass の出力を *ccc.yaml* という名前のファイルに書き込みます。

```
kubectl -n ccc-ns get clusterclass tanzukubernetescluster -o yaml > ccc.yaml
```

または、ショートカット バージョン:

```
kubectl -n ccc-ns get cc tanzukubernetescluster -o yaml > ccc.yaml
```

4 クローン作成された ClusterClass ファイルを編集するために を開きます。

```
vim ccc.yaml
```

5 `ccc.yaml` ファイルを編集します。

- `metadata.creationTimestamp` という行を削除します。
- `metadata.generation` という行を削除します。
- `metadata.resourceVersion` という行を削除します。
- `metadata.uid` という行を削除します。
- `metadata.name` の値を `tanzukubernetescluster` から `ccc` に変更します。
- `metadata.namespace` の値は `ccc-ns` のままにします
- `run.tanzu.vmware.com/resolve-tkr: ""` の値は `metadata.annotations` のままにします。この注釈は TKR データ/解決に必要です。

6 変更を保存して確認します。

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: ClusterClass
metadata:
  annotations:
    run.tanzu.vmware.com/resolve-tkr: ""
  name: ccc
  namespace: ccc-ns
spec:
...
```

7 カスタム ClusterClass オブジェクトを作成します。

```
kubectl apply -f ccc.yaml -n ccc-ns
```

予期される結果：

```
clusterclass.cluster.x-k8s.io/ccc created
```

8 カスタム ClusterClass を一覧表示します。

```
kubectl get cc -n ccc-ns
```

予期される結果：

```
NAME                      AGE
ccc                       3m14s
tanzukubernetescluster    29m
```

2: カスタム ClusterClass に基づく TKG クラスタの作成

クラスタ v1beta1 API を使用して、ClusterClass に基づくクラスタを作成します。

1 `ccc-cluster.yaml` マニフェストを作成してクラスタをプロビジョニングします。

```
#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
```

ここで、

- `metadata.name` の値はクラスタの名前です：`ccc-cluster`
- `spec.topology.class` の値はカスタム ClusterClass の名前です：`ccc`
- `spec.topology.version` の値は TKR バージョンです
- `spec.topology.variables.storageClass` の値はパーシステント ストレージ クラスの名前です

注： テスト目的で、制御プレーンとワーカー ノード プールには 1 つのレプリカで十分です。本番環境では、制御プレーンに 3 つのレプリカを使用し、ワーカー ノード プールごとに少なくとも 3 つのレプリカを使用します。

2 カスタム ClusterClass に基づいて TKG クラスタを作成します。

```
kubectl apply -f ccc-cluster.yaml -n ccc-ns
```

予期される結果：

```
cluster.cluster.x-k8s.io/ccc-cluster created
```

3 クラスタのプロビジョニングを確認します。

次のコマンドを実行します。すべてのクラスタ ノードが正常に起動するまで待機します。

```
kubectl -n ccc-ns get
cc,clusters,vsphereclusters,kcp,machinedeployment,machineset,machine,vspheremachine,virtual
machineservice
```

注： 手順 5 でローリング アップデートの進行状況を監視できるように、このコマンドを別のセッションで実行すると役立ちます。

3: パッケージ更新の有無の確認

ワーカー ノードの 1 つに SSH 接続して、更新するパッケージがあることを確認します。

注： この手順の目的は、実際に更新するパッケージではなく、更新するパッケージがあることを確認することです。新しいクラスタ ノードがロールアウトされると、カスタム ClusterClass によって更新されます(このノードに続く手順)。この手順とその後に続く手順は、カスタム ClusterClass に対して実行できる操作の例を示します。

1 次のコマンドを実行して、SSH シークレットを取得します。

```
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 次のコマンドを実行して、ワーカー ノード仮想マシンの IP アドレスを取得します。

```
kubectl -n ccc-ns get vm -o wide
```

注： 複数のワーカー ノードを展開した場合は、1 つを選択します。制御プレーン ノードは使用しないでください。

3 次のコマンドを実行して、ワーカー ノード仮想マシンに SSH 接続します。

```
ssh -i ${CC}-ssh vmware-system-user@IP-ADDRESS-OF-WORKER-NODE
```

例：

```
ssh -i ${CC}-ssh vmware-system-user@192.168.128.55
```

注： 接続を続行するには、「yes」と入力します。

予期される結果：ホストに SSH 接続すると、次のメッセージが表示されます。

```
tdnf update info not availble yet!
```

4 次のコマンドを実行し、更新を確認します。

```
sudo -i
```

```
tdnf update
```

5 プロンプトで、いいえ(更新しない)に「N」と入力します。

予期される結果：

```
Operation aborted
```

注： ここでの目的は、更新を開始するのではなく、更新の存在を確認するだけです。次のセクションでカスタム ClusterClass にコマンドを追加して、更新を開始します。

6 「exit」と入力して SSH セッションからログアウトし、もう一度「exit」と入力します。

4: カスタム ClusterClass の更新

tdnf 更新を実行する新しいコマンドを使用して、カスタム ClusterClass を更新します。

1 `ccc` という名前のカスタム ClusterClass を編集用に開きます。

```
kubectl edit cc ccc -n ccc-ns
```

2 `postKubeadmCommands` を使用して、次のセクションまで下にスクロールします。

```
  - 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
```

次のコマンドを両方の `valueFrom.template` フィールドに追加します。

```
- tdnf update -y
```

例：

```
  - 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 カスタム ClusterClass への変更を保存し、エディタを閉じます。

```
wq
```

予期される結果：

```
clusterclass.cluster.x-k8s/ccc edited
```

5: クラスタ ノードのローリング アップデートの確認

カスタム ClusterClass を更新すると、その ClusterClass に基づいてプロビジョニングされたクラスタに対して、クラスタ ノードのローリング アップデートがトリガされます。新しいノードには、上記のコマンドが適用されています。

1 次のコマンドを実行して、クラスタがプロビジョニングされていることを確認します。

すべてのクラスタ ノードが正常に起動するまで待機します。

```
kubectl -n ccc-ns get
cc,clusters,vspherecluster,kcp,machinedeployment,machineset,machine,vspheremachine,virtual
machineservice
```

2 新しい UUID を持つ新しいノードがデプロイされたことがわかります。

3 次のコマンドを実行して、ワーカー ノード仮想マシンに SSH 接続します。

```
ssh -i ${CC}-ssh vmware-system-user@IP-ADDRESS-OF-WORKER-NODE
```

予期される結果：ホストに SSH 接続すると、次のメッセージが表示されます。

```
tdnf update info not availble yet!
```

4 次のコマンドを実行します。

```
sudo -i
```

```
tdnf update
```

予期される結果：更新が必要なパッケージが少なくなります。

5 プロンプトで、いいえ(更新しない)に「N」と入力します。

予期される結果：

```
Operation aborted
```

6 次のコマンドを実行して、tdnf が実行されたことを確認します。

```
cat /var/log/cloud-init-output.log | grep -i tdnf
```

7 「exit」と入力して SSH セッションからログアウトし、もう一度「exit」と入力します。

カスタム ClusterClass のメンテナンス

TKG サービス を新しいバージョンにアップグレードした後、以前の TKG サービス バージョンのデフォルトの ClusterClass から派生したカスタム ClusterClass が、新しい TKG サービス バージョンに付属するデフォルトの ClusterClass に対する変更を使用して更新されていることを確認する必要があります。

次のワークフローを使用して、カスタム ClusterClass をシステム提供の ClusterClass と同期させます。これらの手順では、ここに記載されているように、最初のカスタム ClusterClass が作成されていることが前提となります。

1 TKG サービス バージョンをアップグレードします。

たとえば、TKG サービス v3.0 から v3.1 にアップグレードします。

3 章 TKG サービスのインストールとアップグレードを参照してください。

2 ここに記載されているワークフローの概要に従って、新しいカスタム ClusterClass を作成します。

新しいカスタム ClusterClass の名前に TKG サービス のバージョン（`ccc-3.1` など）を追加して、バージョンを手動で管理します。

3 以前のカスタム ClusterClass のカスタム パッチと変数を新しいカスタム ClusterClass に追加します。

この操作を行うには、`cat ccc.yaml` を実行し、その中のカスタム パッチと変数を `ccc-3.1.yaml` にコピーします。

4 新しいカスタム ClusterClass を適用し、調整が成功するまで待機します。

5 クラスタ オブジェクトの `spec.topology.class` フィールドを編集して、以前のカスタム ClusterClass を使用して TKG クラスタを新しいカスタム ClusterClass に更新します。

管理対象外の ClusterClass

vSphere 8 U2 以降では、TKG コントローラでカスタム ClusterClass を管理しない場合、そのカスタム ClusterClass に注釈を追加できます。この注釈を追加する場合は、基盤となるすべての Kubernetes オブジェクト（証明書、シークレットなど）を手動で作成する必要があることに注意してください。その方法については、vSphere 8 U1 カスタム ClusterClass の v1beta1 の例:カスタム ClusterClass に基づくクラスタ(vSphere 8 U1 ワークフロー)を参照してください。

注釈は次のとおりです。

注釈キー	値
run.tanzu.vmware.com/unmanaged-clusterclass	" "

`ccc` という名前のカスタム ClusterClass に注釈を追加する方法の例を次に示します。

```
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 の例:カスタム ClusterClass に基づくクラスタ(vSphere 8 U1 ワークフロー)

カスタム ClusterClass に基づいて TKG クラスタをプロビジョニングするには、次の手順を参照してください。これらの手順は、vSphere 8 U1 環境に固有であることに注意してください。

前提条件

カスタム ClusterClass に基づいて TKG クラスタをプロビジョニングする手順は、vSphere 8 U1 リリース以降で使用できます。vSphere 8 U2 を使用している場合は、「v1beta1 の例：カスタム ClusterClass に基づくクラスタ (vSphere 8 U2 以降のワークフロー)」を参照してください。

次の前提条件を満たす必要があります。

- vSphere 8 U1 環境
- ワークロード管理が有効
- スーパーバイザー 構成済み
- vSphere 向け Kubernetes CLI Tools がインストールされている Ubuntu クライアント

注目： カスタム ClusterClass は、アップストリーム クラスタ API のドキュメントに基づく Kubernetes の試験的な機能です。カスタム ClusterClass で使用可能なカスタマイズの範囲により、VMware は可能なすべてのカスタマイズをテストまたは検証できません。ユーザーは、カスタム ClusterClass クラスタのテスト、検証、およびトラブルシューティングを行う必要があります。カスタム ClusterClass クラスタに関するサポート チケットを発行できますが、VMware サポートはベスト エフォートベースにのみ制限され、カスタム ClusterClass クラスタに対して発行されたすべての問題の解決を保証することはできません。本番環境にカスタム ClusterClass クラスタを展開する前に、これらのリスクを認識しておく必要があります。

パート 1:カスタム ClusterClass の作成

最初のパートでは、`tanzukubernetescluster` という名前のデフォルトの ClusterClass のクローンを作成することで、カスタム ClusterClass を作成します。

1 *custom-ns* という名前の vSphere 名前空間 を作成します。

2 スーパーバイザー にログインします。

3 *custom-ns* という名前の vSphere 名前空間 にコンテキストを切り替えます。

4 デフォルトの ClusterClass を取得します。

```
kubectl get clusterclass tanzukubernetescluster -o json
```

5 デフォルトの ClusterClass のクローンを作成することで、*custom-cc* という名前のカスタム ClusterClass を作成します。

```
kubectl get clusterclass tanzukubernetescluster -o json | jq '.metadata.name="custom-cc"'
| kubectl apply -f -
```

予期される結果：

```
clusterclass.cluster.x-k8s.io/custom-cc created
```

6 カスタム ClusterClass を取得します。

```
kubectl get clusterclass custom-cc -o json
```

必要に応じて、 を使用してカスタム ClusterClass を表示できます。

```
kubectl get clusterclass custom-cc -o json | less
```

注： コマンド「q」を発行して終了を減らします。

パート 2:TKG クラスタをプロビジョニングするために必要な スーパーバイザー オブジェクトの作成

続いてのパートでは、カスタム ClusterClass を使用して、カスタム TKG クラスタの初期デプロイに必要な スーパーバイザー オブジェクトを作成します。

注： デフォルトでは、クラスタ名「ccc-cluster」を使用します。別のクラスタ名を使用している場合は、適切なフィールドを変更する必要があります。

1 自己署名拡張機能証明書の発行者を作成します。

```
#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
```

予期される結果：

```
issuer.cert-manager.io/self-signed-extensions-issuer created
```

2 拡張機能 CA 証明書のシークレットを作成します。

```
#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
```

予期される結果：

```
certificate.cert-manager.io/ccc-cluster-extensions-ca created
```

3 拡張機能 CA 証明書の発行者を作成します。

```
#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
```

予期される結果：

```
issuer.cert-manager.io/ccc-cluster-extensions-ca-issuer created
```

4 認証サービス証明書のシークレットを作成します。

```
#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
```

予期される結果：

```
certificate.cert-manager.io/ccc-cluster-auth-svc-cert created
```

5 発行者と証明書の作成を確認します。

```
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
```

パート 3:カスタム ClusterClass に基づく TKG クラスタの作成

クラスタ v1beta1 API を使用して、ClusterClass に基づくクラスタを作成します。カスタム ClusterClass に基づく v1beta1 クラスタには、次の最小変数セットが必要です。

変数	説明
vmClass	TKG サービス クラスタでの仮想マシン クラスの使用を参照してください。
storageClass	vSphere 名前空間 のパーシステント ストレージの構成を参照してください。
ntp	スーパーバイザー を有効にするために使用される NTP サーバ。
extensionCert	前のセクションで「拡張機能 CA 証明書」が作成された後に自動生成されます。
clusterEncryptionConfigYaml	以下のセクションでは、このファイルを取得するプロセスについて説明します。

1 暗号化シークレットを作成します。

```
#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
```

予期される結果：

```
secret/ccc-cluster-encryption created
```

2 スーパーバイザー から NTP サーバを収集します。

```
kubectl -n vmware-system-vmop get configmap vmoperator-network-config -o
jsonpath={.data.ntpservers}
```

3 `cluster-with-ccc.yaml` マニフェストを作成してクラスタをプロビジョニングします。

```
#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==
```

クラスタ マニフェストで、次のフィールドを確認または更新します。

パラメータ	説明
`metadata.name`	v1beta1 クラスタの名前。
`spec.topology.class`	カスタム ClusterClass の名前。

パラメータ	説明
spec.topology.version	Tanzu Kubernetes リリース のバージョン
spec.topology.variables.storageClass.value	クラスタがプロビジョニングされる vSphere 名前空間 に適用されている StoragePolicy
spec.topology.variables.ntp.value	NTP サーバのアドレス
spec.topology.variables.extensionCert.value.contentSecret.name	確認
spec.topology.variables.clusterEncryptionConfigYaml.value	ClusterEncryptionConfig シークレットの data.key の値を入力します。

4 カスタム ClusterClass に基づいてクラスタを作成します。

```
kubectl apply -f cluster-with-ccc.yaml -n custom-ns
```

予期される結果：

```
cluster.cluster.x-k8s.io/ccc-cluster created
```

vSphere Client を使用して、クラスタが作成されていることを確認します。

5 TKG クラスタにログインします。

```
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
```

パート 4:TKG クラスタを管理するために必要な スーパーバイザー オブジェクトの作成

CCC を使用するクラスタが適用されると、さまざまなコントローラがプロビジョニングを試みます。ただし、基盤となるインフラストラクチャ リソースでは、追加のオブジェクトを適切にブートストラップする必要があります。

パラメータ	値
認証	認証値を収集して values.yaml という名前のファイルに更新する必要があります。
Base64 エンコード	values.yaml ファイルは base64 文字列にエンコードされます。
guest-cluster-auth-service-data-values.yaml	この文字列は、ファイルを適用する前に、 CCC_config_yamls.tar.gz からダウンロードした guest-cluster-auth-service-data-values.yaml ファイルに追加されます。
GuestClusterAuthSvcDataValues シークレット	最後に、ゲスト クラスタ ブートストラップを変更して、新しく作成された GuestClusterAuthSvcDataValues シークレットを参照する必要があります。

1 クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context custom-ns
```

2 authServicePublicKeys の値を取得します。

```
kubectl -n vmware-system-capw get configmap vc-public-keys -o
jsonpath="{.data.vsphere\.local\.json}"
```

結果を values.yaml という名前のテキスト ファイルにコピーします。

```
authServicePublicKeys: '{"issuer_url":"https://...SShrDw=="]}]}}'
```

3 クラスタ UID を取得して、authServicePublicKeys を更新します。

```
kubectl get cluster -n custom-ns ccc-cluster -o yaml | grep uid
```

4 values.yaml ファイルの authServicePublicKeys セクションで、クラスタ UID を「client_id」の値に追加します。

構文：vmware-tes:vc:vns:k8s:*clusterUID*

例：

```
vmware-tes:vc:vns:k8s:7d95b50b-4fd4-4642-82a3-5dbfe87f499c
```

5 証明書の値を取得します（*ccc-cluster* を選択したクラスタ名に置き換えます）。

```
kubectl -n custom-ns get secret ccc-cluster-auth-svc-cert -o jsonpath="{.data.tls\.crt}" |
base64 -d
```

6 証明書を values.yaml に追加します。

authServicePublicKeys セクションの下に証明書の内容を追加します。

注： 障害を回避するには、証明書を 4 つのスペースでインデントする必要があります。

例：

```
authServicePublicKeys: '{"issuer_url":"https://...SShrDw=="]}]}}'
ceritificate: |
    -----BEGIN CERTIFICATE-----
    MIIDPTCCAiWgAwIBAgIQMibGSjeuJelQoPxCof/+xzANBgkqhkiG9w0BAQsFADAg
    ...
    sESk/RDTB1UAvi8PD3zcbEKZuRxuo4IAJqFFbAabwULhjUo0UwT+dIJo1gLf5/ep
    VoIRJS7j6VT98WbKyZp5B4I=
    -----END CERTIFICATE-----
```

7 privateKey の値を取得します。

```
kubectl -n custom-ns get secret ccc-cluster-auth-svc-cert -o jsonpath="{.data.tls\.key}"
```

8 values.yaml ファイルを確認します。

```
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 values.yaml ファイルを base64 エンコードでハッシュ化して、guest-cluster-auth-service-data-values.yaml ファイルの出力を収集します。

```
base64 -i values.yaml -w 0
```

10 guest-cluster-auth-service-data-values.yaml ファイルを作成します。

シークレットのテンプレートを次に示します。

```
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
```

次の表を参照して、想定されるシークレット値を入力します。

パラメータ	値
data.values.yaml	values.yaml の base64 エンコード文字列
metadata.labels.cluster-name	クラスタの名前（例：ccc-cluster）
metadata.labels.package-name	guest-cluster-auth-service.tanzu.vmware.com.*version* この値を取得するには、kubectl get tkr v1.26.5---vmware.2-fips.1-tkg.1 -o yaml コマンドを実行します 使用しているバージョンに応じて TKR のバージョンを変更します。
metadata.name	クラスタの名前（例：ccc-cluster）

11 guest-cluster-auth-service-data-values.yaml シークレットを作成します。

```
kubectl apply -f guest-cluster-auth-service-data-values.yaml -n custom-ns
```

12 クラスタ ブートストラップを編集してシークレットを参照します。

```
kubectl edit clusterbootstrap ccc-cluster -n custom-ns
```

13 `guest-cluster-auth-service.tanzu.vmware.com.version:` という行の下に次の行を追加します。

```
valuesFrom:
  secretRef: ccc-cluster-guest-cluster-auth-service-data-values
```

例：

```
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 保存して終了し、clusterbootstrap の変更を適用します。

パート 5:ポッド セキュリティの構成

TKR バージョン 1.25 以降を使用している場合は、`custom-ns` という名前の vSphere 名前空間 のポッド セキュリティを構成します。TKR 1.25 以降の PSA の構成を参照してください。

TKR バージョン 1.24 以前を使用している場合、クラスタ内のポッドはポッド セキュリティ ポリシーへのバインドを必要とします。必要なリソース オブジェクトをクラスタ レベルで適用するには、次のプロセスを使用します。

1 TKG クラスタ kubeconfig を収集します。

```
kubectl -n custom-ns get secret ccc-cluster-kubeconfig -o jsonpath="{.data.value}" |
base64 -d > ccc-cluster-kubeconfig
```

2 `psp.yaml` ファイルを作成します。

```
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 ポッドのセキュリティ ポリシーを適用します。

```
KUBECONFIG=ccc-cluster-kubeconfig kubectl apply -f psp.yaml
```

4 TKG クラスタにログインします。

```
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 名前空間を一覧表示します。

```
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
```

パート 6:vSphere SSO ロールとカスタム TKG クラスタの同期

開発者がクラスタ ワークロードを管理するには、vSphere 名前空間 に組み込まれた vCenter Single Sign-On ユーザーのロールバインドを スーパーバイザー から TKG クラスタに同期する必要があります。

このプロセスでは、既存のロールバインド リストを スーパーバイザー からエクスポートし、「編集」ロールを持つロールバインドを収集し、`sync-cluster-edit-rolebinding.yaml` ファイルを作成してから、KUBECONFIG を使用して TKG クラスタに適用する必要があります。

1 スーパーバイザー から既存のロールバインドを収集します。

```
kubectl get rolebinding -n custom-ns  -o yaml
```

2 ロールバインド オブジェクトの返されたリストから、「編集」と等しい `roleRef.name` を持つオブジェクトを特定します。

例：

```
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 `sync-cluster-edit-rolebinding.yaml` という名前のファイルを作成して、デフォルトの administrator@vsphere.local 以外のロールバインドを追加します。例：

```
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
```

注： `metadata.name` フィールドでは、すべてのユーザーのユーザー ロールの先頭に `vmware-system-auth-sync-` が付加されます。metadata.name エントリと subjects.name エントリには、デフォルト以外のすべてのロールの変更が必要です。

4 sync-cluster-edit-rolebinding.yaml 構成を適用して、ロールバインドを同期します。

```
KUBECONFIG=ccc-cluster-kubeconfig kubectl apply -f sync-cluster-edit-rolebinding.yaml
```

TanzuKubernetesCluster v1alpha3 API の使用

このセクションでは、v1alpha3 API を使用して `TanzuKubernetesCluster` タイプの TKG サービス クラスタをプロビジョニングするためのリファレンス コンテンツを提供します。必要に応じてさまざまな構成やカスタマイズを行う例も含まれています。

TanzuKubernetesCluster v1alpha3 API

v1alpha3 API を使用すると、スーパーバイザー の TKG を使用して TanzuKubernetesCluster をプロビジョニングできます。v1alpha3 API のドキュメントについては、このトピックを参照してください。

TanzuKubernetesCluster v1alpha3 API

この仕様には、v1alpha3 API を使用して `TanzuKubernetesCluster` をプロビジョニングする際に使用できるすべてのパラメータがリストされています。

重要： 有効なキー名は、英数字、ダッシュ（`key-name` など）、アンダースコア（`KEY_NAME` など）、ドット（`key.name` など）のみで構成する必要があります。キー名にスペース文字を使用することはできません。

```
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
```

TanzuKubernetesCluster v1alpha3 API – 注釈付き

この注釈付きの仕様には、v1alpha3 API を使用して TanzuKubernetesCluster をプロビジョニングする際に使用できるすべてのパラメータと、各フィールドのドキュメントがリストされています。

```
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 の例：デフォルトの TanzuKubernetesCluster

v1alpha3 API を使用してデフォルトの TanzuKubernetesCluster をプロビジョニングするには、サンプル YAML を参照してください。

v1alpha3 の例：デフォルトの TanzuKubernetesCluster

このサンプル YAML は、v1alpha3 API を使用してデフォルトの TanzuKubernetesCluster をプロビジョニングします。

この例は、TKC のプロビジョニングに必要な最小構成を示しています。デフォルトのネットワークおよびストレージの設定が使用されるため、YAML から除外されます。制御プレーン ノードとワーカー ノードの両方に、参照されている TKR が使用されます。

```
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 の例：デフォルトのストレージとノード ボリュームを使用する TKC

ノード ボリュームにデフォルトのストレージ クラス設定とカスタム設定および v1alpha3 API を使用して TanzuKubernetesCluster をプロビジョニングするには、サンプル YAML を参照してください。

v1alpha3 の例：ノード ボリュームとデフォルト ストレージを使用する TKC

このサンプル YAML は、v1alpha3 API を使用してカスタムの TanzuKubernetesCluster をプロビジョニングします。

この例では、次のオプションのカスタマイズに注目してください。詳細については、TanzuKubernetesCluster v1alpha3 API を参照してください。

- クラスタは、Helm や Tanzu パッケージによってデプロイされたワークロードなど、一部のツールで必要になるデフォルトのストレージ クラスを使用してプロビジョニングされます
- ワーカー ノード ボリュームは、`containerd` や `kubelet` など、変更の多いコンポーネントに対して宣言されます

```
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 の例：カスタム ネットワークを使用する TKC

カスタム ネットワーク設定で v1alpha3 API を使用して TanzuKubernetesCluster をプロビジョニングするには、サンプル YAML を参照してください。

v1alpha3 の例：カスタム ネットワーク設定を使用する TKC

ネットワークは次のようにカスタマイズされています。詳細については、TanzuKubernetesCluster v1alpha3 API を参照してください。

- デフォルトの Antrea ではなく、Calico CNI が使用される
- ポッドおよびサービスにデフォルト以外のサブネットが使用される
- プロキシ サーバと TLS 証明書が宣言される

```
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==
```

TKC ポッド ネットワークのカスタマイズに関する考慮事項

クラスタ仕様の設定 `spec.settings.network.pods.cidrBlocks` のデフォルトは 192.168.0.0/16 です。

カスタマイズする場合、ポッドの CIDR ブロックの最小サイズは /24 です。ただし、/16 を超える `pods.cidrBlocks` サブネット マスクを制限する場合は注意が必要です。

TKG は、`pods.cidrBlocks` から取得された /24 サブネットを各クラスタ ノードに割り当てます。この割り当てを指定するには、Kubernetes Controller Manager を選択し、NodeCIDRMaskSize という名前の NodeIPAMController パラメータを指定して、クラスタ内のノード CIDR のサブネット マスク サイズを設定します。IPv4 のデフォルトのノード サブネット マスクは /24 です。

クラスタ内の各ノードは `pods.cidrBlocks` から /24 サブネットを取得するため、プロビジョニングするクラスタに対して制限が厳しすぎるサブネット マスク サイズを使用すると、ノードの IP アドレスが不足する可能性があります。

Antrea または Calico CNI でプロビジョニングされた Tanzu Kubernetes クラスタには、次のノード制限が適用されます。

/16 == 最大 150 ノード（ConfigMax あたり）

/17 == 最大 128 ノード

/18 == 最大 64 ノード

/19 == 最大 32 ノード

/20 == 最大 16 ノード

/21 == 最大 8 ノード

/22 == 最大 4 ノード

/23 == 最大 2 ノード

/24 == 最大 1 ノード

v1alpha3 の例：Ubuntu TKR を使用する TKC

クラスタ ノードに Ubuntu オペレーティング システムを使用する TanzuKubernetesCluster クラスタをプロビジョニングするには、こちらのサンプル YAML を参照してください。このようなクラスタを、vGPU ワークロードに使用できます。

v1alpha3 の例：TKC と Ubuntu TKR を使用する TKC

デフォルトでは、TKG クラスタ ノードには、指定した TKR の PhotonOS エディションが使用されます。参照されている TKR で OSImage 形式がサポートされ、Ubuntu OS エディションが使用可能な場合は、`run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu` 注釈を使用して TKR の Ubuntu OS エディションを指定します。OSImage 形式の詳細については、TKr オペレーティング システムのイメージ形式を参照してください。

AI/ML ワークロードには Ubuntu TKR が必要です。各ワーカー ノードのプールには、コンテナ化されたランタイムと kubelet の専用ボリュームがあり、それぞれ 70 GiB の容量があります。コンテナベースの AI/ML ワークロードには、このサイズの別のボリュームを使用することを推奨します。

```
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 の例：複数の vSphere Zone を横断する TKC

v1alpha3 API を使用して、複数の vSphere Zone を横断する TanzuKubernetesCluster をプロビジョニングするには、サンプル YAML を参照してください。

vSphere Zone と障害ドメイン

vSphere Zone を使用すると、可用性の高い TKG クラスタを スーパーバイザー に作成できます。複数の vSphere Zone を横断する TKG クラスタをプロビジョニングする場合は、各ノード プールに障害ドメインを指定する必要があります。

各障害ドメインは vSphere Zone にマッピングされ、vSphere Zone は 1 つの vSphere クラスタに関連付けられます。vSphere フォルト ドメインとも呼ばれる障害ドメインは、vSphere Zone を作成するときに、vSphere 管理者によって定義および管理されます。TKG クラスタに使用するストレージ プロファイルは、`zonal` として構成する必要があります。TKG サービス クラスタの vSphere ストレージ ポリシーの作成を参照してください。

レプリカを持つポッドを スーパーバイザー 上の TKG クラスタにデプロイすると、ポッド インスタンスは vSphere Zone 全体に自動的に分散されます。TKG クラスタにポッドをデプロイするときにゾーンの詳細を指定する必要はありません。

TKG 環境で vSphere Zone の可用性を確認するには、TKG クラスタをプロビジョニングする vSphere 名前空間から次のいずれかのコマンドを実行します。

```
kubectl get vspherezones
```

```
kubectl get availabilityzones
```

`system:authenticated` ユーザーは、両方のコマンドを利用できます。vSphere Zones は スーパーバイザー スコープのリソースであるため、名前空間を指定する必要はありません。

v1alpha3 の例：複数の vSphere Zone を横断する TKC

このサンプル YAML では、複数の vSphere Zone を横断する TKG クラスタをプロビジョニングします。

このサンプルでは、各 nodePool の `failureDomain` パラメータで vSphere Zone を指定しています。このパラメータの値は、vSphere Zone の名前です。

```
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 の例：ルーティング可能なポッド ネットワークを使用する TKC

ルーティング可能なポッド ネットワークを使用する TanzuKubernetesCluster を作成するには、クラスタの CNI として `antrea-nsx-routed` を指定して、ルーティング可能な名前空間ネットワークを スーパーバイザー に構成します。

ルーティング可能なポッド ネットワークについて

`antrea` または `calico` CNI プラグインを使用して Tanzu Kubernetes クラスタをプロビジョニングすると、デフォルトのポッド ネットワーク `192.168.0.0/16` が作成されます。このサブネットは、クラスタ内でのみ一意のプライベート アドレス空間であり、ネットワーク上ではルーティングできません。

TKG v1alpha3 API は、`antrea-nsx-routed` CNI プラグインを使用してルーティング可能なポッド ネットワークをサポートします。このネットワーク インターフェイスは、カスタマイズされた Antrea プラグインであり、TKG クラスタでルーティング可能なポッド ネットワークをサポートするように構成されています。クラスタの仕様では、IP アドレス管理 (IPAM) が スーパーバイザー によって処理されるようにするために、ポッドの CIDR ブロック フィールドを明示的に null にする必要があります。次の例を参照してください。

ルーティング可能なポッド ネットワークを有効にすると、クラスタ外のクライアントから直接ポッドのアドレスを指定できます。また、外部のネットワーク サービスおよびサーバがソース ポッドを識別し、IP アドレスに基づいてポリシーを適用できるように、ポッドの IP アドレスが保存されます。次のようにトラフィック パターンがサポートされます。

- 同じ vSphere 名前空間 にある TKG クラスタ ポッドと vSphere ポッド の間では、トラフィックが許可されます。
- 異なる vSphere 名前空間 にある TKG クラスタ ポッドと vSphere ポッド の間では、トラフィックがドロップされます。
- スーパーバイザー 制御プレーン ノードは、TKG クラスタ ポッドにアクセスできます。

- TKG クラスタ ポッドは外部ネットワークにアクセスできます。
- 外部ネットワークは、TKG クラスタ ポッドにアクセスできません。クラスタ ノードでは、分散ファイアウォール (DFW) の隔離ルールによってトラフィックがドロップされます。

ルーティング可能なポッド ネットワークの作成：スーパーバイザー 構成

ルーティング可能なポッド ネットワークを作成するには、スーパーバイザー と TKG クラスタで構成を行う必要があります。

注： ルーティング可能なポッド ネットワークを使用するには、NSX に スーパーバイザー を構成する必要があります。VDS ネットワークでは、ルーティング可能なポッドを使用することはできません。

ルーティング可能なポッド ネットワークを スーパーバイザー で構成するには、次の手順を実行します。

1 新しい vSphere 名前空間 を作成します。

TKG サービス クラスタをホストするための vSphere 名前空間 の作成を参照してください。

2 [スーパーバイザー ネットワーク設定のオーバーライド] チェックボックスをオンにします。

詳細については、vSphere 名前空間のワークロード ネットワーク設定のオーバーライドを参照してください。

3 ルーティング可能なポッド ネットワークを次のように構成します。

フィールド	説明
[NAT モード]	ルーティング可能なサブネットを使用しているためにネットワーク アドレス変換 (NAT) を無効にするには、このオプションを選択解除します。
[名前空間ネットワーク CIDR]	名前空間ネットワーク CIDR は、vSphere 名前空間の IP アドレス プールとして動作するサブネットです。名前空間サブネット プリフィックスには、その IP アドレス プールから分割された後続の CIDR ブロックのサイズが記述されます。 このフィールドには、ルーティング可能な IP サブネットが IP アドレス/ビット（例：10.0.0.6/16）の形式でポピュレートされます。NCP によって、ネットワークに指定された IP ブロックから 1 つ以上の IP プールが作成されます。 少なくともサブネット サイズ /23 を指定する必要があります。たとえば、サブネット プリフィックスが /28 のルーティング可能なサブネット /23 を指定すると、6 ノード クラスタに十分な数である 32 個のサブネットが取得されます。プリフィックスが /28 の /24 サブネットを指定した場合に取得されるサブネットは 2 個であり、これでは不十分です。
[名前空間サブネット プリフィックス]	名前空間サブネット プリフィックスには、名前空間ネットワークの IP アドレス プールから分割された後続の CIDR ブロックのサイズが記述されます。 たとえば、/28 の形式でサブネット プリフィックスを指定します。

4 [作成] をクリックして、ルーティング可能なポッド ネットワークを作成します。

ルーティング可能なポッド ネットワークの作成：TKG クラスタの構成

次のサンプル YAML は、ルーティング可能なポッド ネットワークを使用してクラスタを構成する方法を示します。

クラスタの仕様では、ルーティング可能なポッド ネットワークを有効にするために antrea-nsx-routed を CNI として宣言しています。antrea-nsx-routed が指定されている場合、NSX-T ネットワークが使用されていないとクラスタのプロビジョニングは失敗します。

CNI が antrea-nsx-routed と指定されている場合、pods.cidrBlock フィールドを空にする必要があります。

```
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 の例：SSL/TLS 用の追加の信頼できる CA 証明書を含む TKC

SSL/TLS 用の追加の信頼できる CA 証明書を含む v1alpha3 API を使用して TanzuKubernetesCluster をプロビジョニングするには、YAML の例を参照してください。

v1alpha3 の例：追加の信頼できる CA 証明書を含む TKC

クラスタは次のようにカスタマイズされます。詳細については、TanzuKubernetesCluster v1alpha3 API を参照してください。

- 追加の信頼できる CA 証明書がクラスタ仕様の `network.trust.additionalTrustedCAs` セクションで宣言されています
- `additionalTrustedCAs` フィールドは、次に示す名前と値のペアの配列です。
  - `name` フィールドはユーザー定義の文字列です
  - `data` 値は、base64 エンコードの PEM 形式の CA 証明書の内容です

```
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==
```

手順：新しいクラスタ

新しい TKGS クラスタに追加の信頼できる CA 証明書を 1 つ以上含める場合は、次の手順を実行します。

1 1 つ以上の CA 証明書の名前とデータ値を `additionalTrustedCAs` フィールドにポピュレートします。

2 通常どおりクラスタをプロビジョニングします。

Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。

3 クラスタが正常にプロビジョニングされると、追加した CA 証明書がクラスタによって信頼されます。

手順：既存のクラスタ

既存のクラスタに追加の信頼できる CA 証明書を 1 つ以上追加する場合は、次の手順を実行します。

1 kubectl の編集が構成されていることを確認します。

kubectl のテキスト エディタの構成を参照してください。

2 クラスタ仕様を編集します。

```
kubectl edit tanzukubernetescluster/tkgs-cluster-name
```

3 `network.trust.additionalTrustedCAs` セクションを仕様に追加します。

4 1 つ以上の CA 証明書の名前とデータ値を `additionalTrustedCAs` フィールドにポピュレートします。

5 テキスト エディタで変更を保存し、変更が kubectl によって登録されたことを確認します。

```
kubectl edit tanzukubernetescluster/tkgs-cluster-name
tanzukubernetescluster.run.tanzu.vmware.com/tkgs-cluster-name edited
```

6 クラスタのローリング アップデートが開始されると、信頼できる CA 証明書が追加されます。

TKG サービス クラスタのローリング アップデート モデルについてを参照してください。

追加の信頼できる CA 証明書の確認

クラスタに追加された信頼できる CA 証明書は、クラスタの kubeconfig ファイルに含まれています。

追加の信頼できる CA 証明書のトラブルシューティング

追加の信頼できる CA に関するエラーのトラブルシューティングを参照してください。

使用事例

最も一般的な使用事例は、コンテナ レジストリに接続するための信頼できる CA を追加する場合です。TKG サービス クラスタとプライベート コンテナ レジストリの統合を参照してください。

TKG サービス クラスタの操作

8

このセクションには、TKG サービス クラスタの操作に関するトピックが記載されています。

次のトピックを参照してください。

- kubectl のテキスト エディタの構成
- kubectl を使用したクラスタの手動スケーリング
- vSphere Client を使用した TKG クラスタのステータスの監視
- kubectl を使用した TKG クラスタのステータスの監視
- kubectl を使用した TKG クラスタの準備の確認
- kubectl を使用した TKG クラスタのマシンの健全性の確認
- kubectl を使用した TKG クラスタの健全性の確認
- kubectl を使用した TKG クラスタのボリュームの健全性の確認
- Tanzu Kubernetes Grid クラスタでのボリュームの健全性の監視
- vSphere Client を使用したパーシステント ボリュームの監視
- Kubectl を使用した TKG クラスタ シークレットの取得
- Kubectl を使用した TKG クラスタ ネットワークの確認
- kubectl を使用した TKG クラスタの操作の確認
- TKG クラスタのライフサイクル ステータスの表示
- kubectl を使用した TKG クラスタのリソース階層の表示
- v1beta1 クラスタの MachineHealthCheck の構成

kubectl のテキスト エディタの構成

TKG クラスタを運用および維持するには、kubectl のデフォルトのテキスト エディタを構成します。

kubectl edit コマンドの使用

TKG クラスタをプロビジョニングしたら、クラスタを運用および維持します。一般的なタスクには、クラスタ ノードのスケーリングと TKR バージョンの更新があります。これらのタスクを実行するには、kubectl edit コマンドを使用してクラスタ マニフェストを更新します。

`kubectl edit CLUSTER-KIND/CLUSTER-NAME` コマンドを実行すると、KUBE_EDITOR または EDITOR 環境変数によって定義されたテキスト エディタでクラスタ マニフェストが開きます。マニフェストの変更を保存すると、`kubectl` から編集内容が正常に記録されたことが報告され、この変更内容を使用してクラスタが更新されます。

例：

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkg-cluster-1 edited
```

変更をキャンセルするには、保存せずにエディタを閉じます。

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
Edit cancelled, no changes made.
```

kubectl edit の構成

`kubectl edit` コマンドを使用するために、Linux では EDITOR 環境変数が設定されています。そうでない場合は、KUBE_EDITOR 環境変数を作成し、変数の値として優先するテキスト エディタを指定します。行った変更をコミット（保存）したときに kubectl が認識できるように、監視フラグ (`-w`) を追加します。

使用しているオペレーティング システムの手順を参照してください。

Linux

Linux（Ubuntu など）で `kubectl edit` を構成する場合、デフォルトのコマンドライン `EDITOR` は Vim です。この場合、`kubectl edit` コマンドを使用するための追加のアクションは不要です。

別のテキスト エディタを使用する場合は、`KUBE_EDITOR` という名前の環境変数を作成して、値を優先テキスト エディタのパスに設定します。

Mac OS

Mac OS で `kubectl edit` を構成する場合は、`KUBE_EDITOR` という名前の環境変数を作成して、値を優先テキスト エディタのパスに設定します。行った変更をコミット（保存）したときにエディタが認識できるように、待機フラグ（ `--wait`、またはショートカット `-w`）を値に追加します。

たとえば、次のように `.bash_profile` に追加すると、kubectl のデフォルトのテキスト エディタとして Sublime が設定され、変更を保存したときにエディタが認識できるように待機フラグが追加されます。

```
export KUBE_EDITOR="/Applications/Sublime.app/Contents/SharedSupport/bin/subl -w"
```

Windows

Windows で `kubectl edit` を構成する場合は、`KUBE_EDITOR` という名前のシステム環境変数を作成して、値を優先テキスト エディタのパスに設定します。この値に監視フラグ (`-w`) を追加します。

たとえば、次の環境変数は、kubectl のデフォルトのテキスト エディタとして Visual Studio Code を設定し、変更を保存したときに Kubernetes が認識できるように監視フラグを追加します。

```
KUBE_EDITOR=code -w
```

Sublime を Windows の kubectl エディタとして構成するには、システム パスに Sublime プログラム ディレクトリを追加して、Sublime 実行ファイルのシステム変数を作成します。例：

システム パスの追加：

```
C:\Program Files\Sublime Text 3\
```

システム変数名と値：

```
KUBE_EDITOR=sublime_text.exe -w
```

kubectl を使用したクラスタの手動スケーリング

ノード数を変更すると、TKG サービス クラスタを水平方向にスケーリングできます。また、ノードをホストする仮想マシン クラスを変更すると、垂直方向にスケーリングできます。クラスタ ノードに接続されたボリュームをスケーリングすることもできます。

サポートされている手動スケーリング操作

次の表に、TKG クラスタでサポートされているスケーリング操作を示します。

表 8-1. TKGS クラスタでサポートされているスケーリング操作

ノード	水平方向のスケール アウト	水平方向のスケール イン	垂直方向のスケーリング	ボリュームのスケーリング
制御プレーン	はい	なし	はい	はい*
ワーカー	はい	はい	はい	はい

次の考慮事項に留意してください。

- 制御プレーン ノードの数は奇数（1 または 3）である必要があります。制御プレーンのスケール アウトはサポートされていますが、制御プレーンのスケール インはサポートされていません。制御プレーンのスケール アウトを参照してください。
- クラスタ ノードを垂直方向にスケーリングするときに、使用可能なリソースが不足していると、ノード上でワークロードを実行できなくなる可能性があります。そのため、通常は水平方向にスケーリングする方法が推奨されます。ワーカー ノードのスケール アウトを参照してください。
- 仮想マシン クラスは変更できません。TKG クラスタで使用される仮想マシン クラスを編集した後に、このクラスタをスケール アウトした場合、新しいクラスタ ノードは更新されたクラス定義を使用しますが、既存のクラスタ ノードは初期のクラス定義を引き続き使用するため、不一致が生じます。仮想マシン クラスについてを参照してください。
- ワーカー ノードのボリュームはプロビジョニング後に変更できます。また、vSphere 8 U3 以降および互換性のある TKr を使用している場合は、制御プレーン ノードも変更できます。クラスタ ノード ボリュームのスケーリングを参照してください。

スケーリングの前提条件：Kubectl 編集の構成

TKG クラスタをスケーリングするには、`kubectl edit CLUSTER-KIND/CLUSTER-NAME` コマンドを使用してクラスタのマニフェストを更新します。マニフェストに対する変更を保存すると、変更が反映されてクラスタが更新されます。kubectl のテキスト エディタの構成を参照してください。

例：

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkg-cluster-1 edited
```

変更をキャンセルするには、保存せずにエディタを閉じます。

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
Edit cancelled, no changes made.
```

制御プレーンのスケール アウト

制御プレーン ノードの数を 1 から 3 に増やすことで、TKG クラスタをスケール アウトします。

注： 本番クラスタには 3 台の制御プレーン ノードが必要です。

1 スーパーバイザー にログインします。

```
kubectl vsphere login --server=SUPERVISOR-IP-ADDRESS --vsphere-username USERNAME
```

2 TKG クラスタが実行されている vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context tkg-cluster-ns
```

3 vSphere 名前空間 で実行されている Kubernetes クラスタを一覧表示します。

次の構文を使用します。

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

たとえば、v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

たとえば、v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get cluster -n tkg-cluster-ns
```

4 ターゲット クラスタで実行されているノードの数を取得します。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

TKG クラスタには、1 台の制御プレーン ノードと、3 台のワーカー ノードが含まれています。

```
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
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get cluster tkg-cluster-1
```

5 `kubectl edit` コマンドを使用して、編集するクラスタ マニフェストをロードします。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl edit cluster/tkg-cluster-1
```

クラスタ マニフェストは、KUBE_EDITOR または EDITOR 環境変数によって定義されたテキスト エディタで開かれます。

6 マニフェストの `spec.topology.controlPlane.replicas` セクションで、制御プレーン ノードの数を 1 から 3 に増やします。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
...
spec:
  topology:
    controlPlane:
      replicas: 1
...
```

```
...
spec:
  topology:
    controlPlane:
      replicas: 3
...
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
...
spec:
  ...
  topology:
    class: tanzukubernetescluster
    controlPlane:
```

```
      metadata: {}
      replicas: 1
    variables:
...
```

```
...
spec:
  ...
  topology:
    class: tanzukubernetescluster
    controlPlane:
      metadata: {}
      replicas: 3
    variables:
...
```

7 テキスト エディタにファイルを保存して、変更を適用します。(キャンセルするには、保存せずにエディタを閉じます。)

マニフェストに対する変更を保存すると、kubectl によって変更がクラスタに適用されます。バックグラウンドでは、スーパーバイザー の 仮想マシン サービス によって新しい制御プレーン ノードがプロビジョニングされます。

8 新しいノードが追加されていることを確認します。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

スケール アウト制御プレーンのノード数が 3 になりました。

```
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
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get cluster tkg-cluster-1
```

ワーカー ノードのスケール アウト

ワーカー ノードの数を増やすことで、TKG クラスタをスケール アウトできます。

1 スーパーバイザー にログインします。

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

2 TKG クラスタが実行されている vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context tkg-cluster-ns
```

3 vSphere 名前空間 で実行されている Kubernetes クラスタを一覧表示します。

次の構文を使用します。

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

たとえば、v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

たとえば、v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get cluster -n tkg-cluster-ns
```

4 ターゲット クラスタで実行されているノードの数を取得します。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

たとえば、次のクラスタには、3 台の制御プレーン ノードと、3 台のワーカー ノードが含まれています。

```
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
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get cluster tkg-cluster-1
```

5 `kubectl edit` コマンドを使用して、編集するクラスタ マニフェストをロードします。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl edit cluster/tkg-cluster-1
```

クラスタ マニフェストは、KUBE_EDITOR または EDITOR 環境変数によって定義されたテキスト エディタで開かれます。

6 ターゲット ワーカー ノード プールの `spec.topology.nodePools.NAME.replicas` 値を編集して、ワーカー ノードの数を増やします。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
...
spec:
  topology:
    ...
    nodePools:
    - name: worker-1
      replicas: 3
...
```

```
...
spec:
  topology:
    ...
    nodePools:
    - name: worker-1
      replicas: 4
...
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
...
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 変更を適用するには、テキスト エディタでこのファイルを保存します。変更をキャンセルするには、保存せずにエディタを閉じます。

ファイルを保存すると、kubectl によって変更がクラスタに適用されます。バックグラウンドでは、スーパーバイザー の 仮想マシン サービス によって新しいワーカー ノードがプロビジョニングされます。

8 新しいノードが追加されていることを確認します。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

スケール アウト後、クラスタ内のワーカー ノード数は 4 になります。

```
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
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get cluster tkg-cluster-1
```

ワーカー ノードのスケール イン

ワーカー ノードの数を減らすことで、TKG クラスタをスケール インできます。

1 スーパーバイザー にログインします。

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

2 TKG クラスタが実行されている vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context tkg-cluster-ns
```

3 vSphere 名前空間 で実行されている Kubernetes クラスタを一覧表示します。

次の構文を使用します。

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

たとえば、v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

たとえば、v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get cluster -n tkg-cluster-ns
```

4 ターゲット クラスタで実行されているノードの数を取得します。

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

5 ターゲット クラスタで実行されているノードの数を取得します。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

たとえば、次のクラスタには、3 台の制御プレーン ノードと、4 台のワーカー ノードが含まれています。

```
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
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get cluster tkg-cluster-1
```

6 `kubectl edit` コマンドを使用して、編集するクラスタ マニフェストをロードします。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl edit cluster/tkg-cluster-1
```

クラスタ マニフェストは、KUBE_EDITOR または EDITOR 環境変数によって定義されたテキスト エディタで開かれます。

7 ターゲット ワーカー ノード プールの `spec.topology.nodePools.NAME.replicas` 値を編集して、ワーカー ノードの数を減らします。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
...
spec:
  topology:
    ...
```

```
    nodePools:
    - name: worker-1
      replicas: 4
...
```

```
...
spec:
  topology:
    ...
    nodePools:
    - name: worker-1
      replicas: 3
...
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
...
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 変更を適用するには、テキスト エディタでこのファイルを保存します。変更をキャンセルするには、保存せずにエディタを閉じます。

ファイルを保存すると、kubectl によって変更がクラスタに適用されます。バックグラウンドでは、スーパーバイザー の 仮想マシン サービス によって新しいワーカー ノードがプロビジョニングされます。

9 ワーカー ノードが削除されたことを確認します。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get tanzukubernetescluster tkg-cluster-1
```

スケール イン後、クラスタ内のワーカー ノード数は 3 になります。

```
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
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get cluster tkg-cluster-1
```

クラスタの垂直方向のスケーリング

スーパーバイザー 上の TKG は、クラスタ制御プレーンおよびワーカー ノードの垂直方向のスケーリングをサポートしています。クラスタ ノードで使用される TKG サービス クラスタでの仮想マシン クラスの使用を変更して、TKG クラスタを垂直方向にスケーリングします。TKG クラスタがプロビジョニングされている vSphere 名前空間 に、使用する仮想マシン クラスをバインドする必要があります。

スーパーバイザー 上の TKG は、システムに組み込まれているローリング アップデート メカニズムを介した垂直方向のスケーリングをサポートしています。`VirtualMachineClass` の定義を変更すると、この新しいクラスを使用して新しいノードがロール アウトされて、古いノードがスピン ダウンされます。9 章 TKG サービス クラスタの更新を参照してください。

1 スーパーバイザー にログインします。

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

2 TKG クラスタが実行されている vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context tkg-cluster-ns
```

3 vSphere 名前空間 で実行されている Kubernetes クラスタを一覧表示します。

次の構文を使用します。

```
kubectl get CLUSTER-KIND -n tkg-cluster-ns
```

たとえば、v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get tanzukubernetescluster -n tkg-cluster-ns
```

たとえば、v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl get cluster -n tkg-cluster-ns
```

4 ターゲット TKG クラスタを記述し、仮想マシンのクラスを確認します。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl describe tanzukubernetescluster tkg-cluster-1
```

たとえば、次のクラスタでは best-effort-medium 型の仮想マシン クラスが使用されています。

```
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: best-effort-medium
      ...
    nodePools:
    - name: worker-nodepool-a1
      replicas: 3
      vmClass: best-effort-medium
      ...
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl describe cluster tkg-cluster-1
```

たとえば、次のクラスタでは best-effort-medium 型の仮想マシン クラスが使用されています。

```
...
Topology:
      ...
    Variables:
      ...
      Name:   vmClass
      Value:  best-effort-medium
    ...
```

注： v1beta1 API クラスタの場合、デフォルトでは vmClass は単一の変数としてグローバルに設定されます。この設定をオーバーライドして、制御プレーン ノードとワーカー ノードに別の仮想マシン クラスを使用できます。API リファレンスの「vmClass」を参照してください。

5 使用可能な仮想マシン クラスを一覧表示して、記述します。

```
kubectl get virtualmachineclass
```

```
kubectl describe virtualmachineclass
```

注: 仮想マシン クラスを vSphere 名前空間 にバインドする必要があります。TKG サービス クラスタでの仮想マシン クラスの使用を参照してください。

6 ターゲット クラスタのマニフェスト開いて編集します。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl edit cluster/tkg-cluster-1
```

クラスタ マニフェストは、KUBE_EDITOR または EDITOR 環境変数によって定義されたテキスト エディタで開かれます。

7 仮想マシン クラスを変更して、マニフェストを編集します。

v1alpha3 API クラスタの場合は、制御プレーンの仮想マシン クラスを `guaranteed-medium` に、ワーカー ノードの仮想マシン クラスを `guaranteed-large` に変更します。

```
spec:
  topology:
    controlPlane:
      replicas: 3
      vmClass: guaranteed-medium
      ...
    nodePools:
    - name: worker-nodepool-a1
      replicas: 3
      vmClass: guaranteed-large
      ...
```

v1beta API クラスタの場合は、仮想マシン クラスを `guaranteed-large` に変更します。

```
...
Topology:
      ...
    Variables:
      ...
      Name:   vmClass
      Value:  guaranteed-large
    ...
```

8 変更を適用するには、テキスト エディタでこのファイルを保存します。変更をキャンセルするには、保存せずにエディタを閉じます。

ファイルを保存すると、kubectl によって変更がクラスタに適用されます。バックグラウンドで、スーパーバイザー の TKG は TKG クラスタのローリング アップデートを実行します。

9 TKG クラスタが新しい仮想マシン クラスを使用して更新されていることを確認します。

v1alpha3 API クラスタの場合は、次のコマンドを実行します。

```
kubectl describe tanzukubernetescluster tkg-cluster-1
```

v1beta1 API クラスタの場合は、次のコマンドを実行します。

```
kubectl describe cluster tkg-cluster-1
```

クラスタ ノード ボリュームのスケーリング

ノードの TKG クラスタ仕様では、必要に応じてノードの 1 つ以上のパーシステント ボリュームを宣言できます。ノード ボリュームの宣言は、ワーカー ノードのコンテナ ランタイムや kubelet など、変更の多いコンポーネントに役立ちます。

クラスタの作成後に 1 つ以上のノード ボリュームを追加または変更する場合は、次の考慮事項に留意してください。

ボリューム ノード	説明
ワーカー ノードのボリュームは変更できる	TKG クラスタがプロビジョニングされた後に、ワーカー ノード ボリュームを追加または更新できます。ローリング アップデートを開始すると、クラスタは新しいボリュームまたは変更されたボリュームを使用して更新されます。 **注意:** 新しいボリュームまたは変更されたボリュームを含むワーカー ノードをスケーリングすると、ローリング アップデート中に現在のボリュームのデータが削除されます。次の説明を参照してください。 TKG クラスタ ノード用に宣言されたボリュームは短期ボリュームとして扱われます。TKG クラスタは、vSphere 名前空間内のパーシステント ボリューム要求 (PVC) を使用して、ボリューム容量が TKG クラスタのストレージ割り当てにカウントされるようにします。TKC ボリュームのキャパシティを増やすと、Kubernetes クラスタ API (CAPI) は新しい PVC を使用して新しいワーカーを展開します。この場合、TKG はデータ移行を実行しませんが、Kubernetes はワークロード ポッドを適切にスケジューリングまたは再スケジューリングします。
制御プレーン ノードのボリュームは変更できる（vSphere 8 U3 以降を使用している場合）	vSphere 8 U3 以降および互換性のある Tanzu Kubernetes リリースを使用している場合は、TKG サービス クラスタがプロビジョニングされた後に、制御プレーン ノードのボリュームを追加または更新できます。 vSphere 8 U3 以降を使用していない場合、Kubernetes クラスタ API (CAPI) では、クラスタの作成後に `spec.toplogy.controlPlane.volumes` を変更することはできません。 クラスタの作成後に制御プレーン ボリュームを追加または変更しようとすると、要求は拒否され、「変更できないフィールドの更新は許可されていません」というエラー メッセージが表示されます。

宣言されたノード ボリュームを含む、v1alpha3 API に基づくクラスタ仕様の抜粋を次に示します。必要に応じて、この抜粋の取得元になる TKG クラスタの完全な例については、「v1alpha3 の例：デフォルトのストレージとノード ボリュームを使用する TKC」を参照してください。v1beta1 API クラスタの例については、「v1beta1 の例：デフォルトの ClusterClass に基づくカスタム クラスタ」を参照してください。

```
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:
      ...
```

vSphere Client を使用した TKG クラスタのステータスの監視

vSphere Client を使用して、TKG クラスタのステータスを監視できます。

手順

1 vSphere Client を使用して、vCenter Server にログインします。

2 [メニュー] の [ホストおよびクラスタ] ビューを選択します。

3 スーパーバイザー が作成されている [データセンター] - [クラスタ] オブジェクトを展開します。

4 [名前空間] リソース プールを展開します。

5 Tanzu Kubernetes クラスタがデプロイされている vSphere 名前空間 を選択します。

各 TKG クラスタは、名前空間リソース プール内のフォルダとして一覧表示されます。各 TKG は、名前の横にある 3 つの六角形のアイコンでグラフィカルに表されます。

6 [メニュー] - [仮想マシンおよびテンプレート] ビューに切り替えます。

クラスタ フォルダ内に、クラスタ ノードを構成する仮想マシンが表示されます。

7 vSphere 名前空間 を選択して、[コンピューティング] タブを選択します。

8 [VMware リソース] で [Tanzu Kubernetes] を選択します。

この vSphere 名前空間 にデプロイされている各 TKG クラスタが一覧表示されます。

kubectl を使用した TKG クラスタのステータスの監視

kubectl を使用して、プロビジョニングされている TKG クラスタのステータスを監視できます。

手順

1 スーパーバイザー で認証します。

2 クラスタが実行されている vSphere 名前空間 に切り替えます。

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 名前空間で実行されている Tanzu Kubernetes クラスタのリストを表示します。

```
kubectl get tanzukubernetesclusters
```

このコマンドは、クラスタのステータスを返します。

4 クラスタの詳細を表示します。

```
kubectl describe tanzukubernetescluster <cluster-name>
```

このコマンドを実行すると、クラスタの詳細が返されます。コマンド出力の [ステータス] セクションに、クラスタに関する詳細情報が表示されます。

```
...
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 追加の `kubectl` コマンドを実行して、クラスタの詳細を表示します。kubectl を使用した TKG クラスタの操作の確認を参照してください。

kubectl を使用した TKG クラスタの準備の確認

TKG コントローラが TKG クラスタをプロビジョニングすると、いくつかのステータス状態が報告されるようになります。これらを使用することでマシンの健全性に関する重要な情報を直接把握できます。

TKG クラスタの準備の確認

TKG クラスタの準備状態を使用して、準備ができていないフェーズまたはコンポーネントがある場合はそれを判別できます。

クラスタの準備を確認したら、vSphereCluster とマシンの状態を使用し、障害を詳細に調べることでさらに診断をすることができます。

TKG クラスタの準備を確認するには、次の手順を実行します。

1 スーパーバイザー にログインします。

2 ターゲット クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。例：

```
kubectl config use-context tkg-cluster-ns
```

3 TKG クラスタのタイプに応じて、`kubectl get tkc -o yaml` コマンドまたは `kubectl get cluster -o yaml` コマンドを実行します。

このコマンドは、クラスタ コンポーネントの準備を返します。準備のさまざまな状態の説明については、以降のセクションを参照してください。

ControlPlaneReady の状態と理由

次の表に、`ControlPlaneReady` の状態の一覧と説明を示します。

表 8-2. ControlPlaneReady の状態

状態のタイプ	説明
`ControlPlaneReady`	制御プレーン ノードがクラスタに対して準備完了し、機能しているかどうかを報告します。

次の表に、`ControlPlaneReady` の状態が false になる理由の一覧と説明を示します。

表 8-3. ControlPlaneReady が False になる理由

理由	説明
WaitingForClusterInfrastructure	クラスタが、ロード バランサなどのマシンの実行に必要な前提条件を待機していることを示します。この理由は、InfrastructureCluster が独自の準備状態を報告していない場合にのみ使用されます。
WaitingForControlPlaneInitialized	最初の制御プレーン ノードが初期化中であることを示します。
WaitingForControlPlane	KubeadmControlPlane の条件を反映します。この理由は、KubeadmControlPlane が独自の準備状態を報告していない場合に使用されます。
クラスタ インフラストラクチャの準備が完了するのを待機している	クラスタが、ネットワークやロード バランサなどのマシンの実行に必要な前提条件を待機していることを示すメッセージ。

NodesHealthy の状態と理由

次の表に、NodesHealthy の状態の一覧と説明を示します。

表 8-4. NodesHealthy の状態

状態のタイプ	説明
NodesHealthy	TankuKubernetesCluster ノードのステータスを報告します。

次の表に、NodesHealthy の状態が true にならない理由の一覧と説明を示します。

表 8-5. NodesHealthy が False になる理由

理由	説明
WaitingForNodesHealthy	すべてのノードが健全ではないドキュメント。

アドオンの状態と理由

次の表に、クラスタ アドオン コンポーネントに関連する状態の一覧と説明を示します。

表 8-6. アドオンの状態

状態のタイプ	説明
AddonsReady	TanzuKubernetesCluster アドオン（CoreDNS、KubeProxy、CSP、CPI、CNI、AuthSvc）の状態のサマリ。
CNIProvisioned	TanzuKubernetesCluster コンテナ ネットワーク インターフェイス (CNI) アドオンのステータスを文書化します。
CSIProvisioned	TanzuKubernetesCluster コンテナ ストレージ インターフェイス (CSI) アドオンのステータスを文書化します。
CPIProvisioned	TanzuKubernetesCluster クラウド プロバイダ インターフェイス (CPI) アドオンのステータスを文書化します。
KubeProxyProvisioned	TanzuKubernetesCluster KubeProxy アドオンのステータスを文書化します。

表 8-6. アドオンの状態 （続き）

状態のタイプ	説明
CoreDNSProvisioned	TanzuKubernetesCluster CoreDNS アドオンのステータスを文書化します。
AuthServiceProvisioned	TanzuKubernetesCluster AuthService アドオンのステータスを文書化します。
PSPProvisioned	PodSecurityPolicy のステータスを文書化します。

次の表に、アドオンの状態が True にならない理由の一覧と説明を示します。警告の原因となっている症状のトラブルシューティングを実行するには、21 章 TKG サービス クラスタのトラブルシューティングを参照してください。

表 8-7. アドオンが False になる理由

理由	重要度	説明
AddonsReconciliationFailed	該当なし	すべてのアドオンの調整に失敗した理由の概要。
CNIProvisioningFailed	警告	CNI アドオンが作成または更新に失敗したことを文書化します。
CSIProvisioningFailed	警告	CSI アドオンが作成または更新に失敗したことを文書化します。
CPIProvisioningFailed	警告	CPI アドオンが作成または更新に失敗したことを文書化します。
KubeProxyProvisioningFailed	警告	KubeProxy アドオンが作成または更新に失敗したことを文書化します。
CoreDNSProvisioningFailed	警告	CoreDNS アドオンが作成または更新に失敗したことを文書化します。
AuthServiceProvisioningFailed	警告	AuthService アドオンが作成または更新に失敗したことを文書化します。
AuthServiceUnManaged		AuthService がコントローラによって管理されていないことを文書化します。
PSPProvisioningFailed	警告	PodSecurityPolicy アドオンが作成または更新に失敗したことを文書化します。

その他の状態と理由

次の表に、StorageClass と RoleBinding の同期、ProviderServiceAccount リソースの調整、ServiceDiscovery、および TKG 2.0 クラスタの互換性の状態の一覧と説明を示します。

表 8-8. その他の状態

条件	説明
StorageClassSynced	スーパーバイザー クラスタからワークロード クラスタへの StorageClass 同期のステータスを文書化します。
RoleBindingSynced	スーパーバイザー クラスタからワークロード クラスタへの RoleBinding 同期のステータスを文書化します。

表 8-8. その他の状態 （続き）

条件	説明
ProviderServiceAccountsReady	プロバイダ サービスのアカウントと、関連するロール、ロールバインド、およびシークレットの作成のステータスを文書化します。
ServiceDiscoveryReady	サービス検出のステータスを文書化します。
TanzuKubernetesReleaseCompatible	TanzuKubernetesCluster が TanzuKubernetesRelease と互換性があるかどうかを示します。

次の表に、その他の状態が True にならない理由の一覧と説明を示します。

表 8-9. その他の理由

理由	説明
StorageClassSyncFailed	StorageClass の同期に失敗したことを報告します。
RoleBindingSyncFailed	ロールバインドの同期に失敗したことを報告します。
ProviderServiceAccountsReconciliationFailed	プロバイダ サービス アカウントに関連するリソースの調整に失敗したことを報告します。
SupervisorHeadlessServiceSetupFailed	スーパーバイザー クラスタ API サーバのヘッドレス サービス設定に失敗したことを文書化します。

kubectl を使用した TKG クラスタのマシンの健全性の確認

TKG コントローラがワークロード クラスタを スーパーバイザー にプロビジョニングすると、いくつかのステータス状態が報告されるようになります。これらを使用することでクラスタの健全性に関する重要な情報を直接把握できます。

マシンの健全性状態について

TKG クラスタは、いくつかの可動部分で構成されます。これらのすべての可動部分は独立した関連性のあるコントローラによって操作され、一連の Kubernetes ノードを構築および維持するために連携します。`TanzuKubernetesCluster` オブジェクトと `Cluster` オブジェクトには、マシンの健全性について詳細な情報を提供するステータス状態があります。

マシンの健全性の確認

TKG クラスタのマシンの健全性を確認するには、次の手順を実行します。

1 スーパーバイザー に接続してログインします。

2 ターゲット TKG クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

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

3 `kubectl describe machine` コマンドを実行します。

このコマンドは、クラスタを構成する仮想マシン ノードの状態を返します。マシンの状態（例：`InfrastructureReady`）が `True` および `Ready` の場合、マシンは健全な状態です。ただし、マシンの状態が `False` になっている場合は、マシンの準備ができていません。それぞれのマシンの状態のタイプの説明については、マシンの健全性状態のリストを参照してください。

4 マシンの準備ができていない場合は、次のコマンドを実行して、インフラストラクチャの問題を特定します。

```
kubectl describe vspheremachine
```

マシンの健全性状態のリスト

次の表に、TKG クラスタで使用可能なマシンの健全性状態の一覧と定義を示します。

条件	説明
`ResourcePolicyReady`	リソース ポリシーの作成が成功したことを報告します。
`ResourcePolicyCreationFailed`	リソース ポリシーの作成中にエラーが発生した場合に報告されます。
`ClusterNetworkReady`	クラスタ ネットワークのプロビジョニングが成功したことを報告します。
`ClusterNetworkProvisionStarted`	クラスタ ネットワークの準備が完了するまで待機している場合に報告されます。
`ClusterNetworkProvisionFailed`	ネットワークのプロビジョニング中にエラーが発生した場合に報告されます。
`LoadBalancerReady`	固定制御プレーン エンドポイントの調整に成功したことを報告します。
`LoadBalancerCreationFailed`	ロードバランサ関連のリソースの作成が失敗した場合に報告されます。
`WaitingForLoadBalancerIP`	ロード バランサの IP アドレスが存在するようになるまで待機している場合に報告されます。
`VMProvisioned`	仮想マシンが作成され、パワーオンされて、IP アドレスが割り当てられていることを報告します。
`WaitingForBootstrapData`	vSphere マシンが、プロビジョニング プロセスを開始する前に、ブートストラップ スクリプトの準備ができるまで待機している場合に報告されます。
`VMCreationFailed`	仮想マシンの CRD または対応するブートストラップ ConfigMap の作成が失敗したことを報告します。
`VMProvisionStarted`	仮想マシンが現在、作成プロセスにある場合に報告されます。
`PoweringOn`	仮想マシンが現在パワーオン シーケンスを実行している場合に報告されます。
`WaitingForNetworkAddress`	マシンのネットワーク設定がアクティブになるまで待機している場合に報告されます。
`WaitingForBIOSUUID`	マシンに BIOS UUID が設定されるまで待機している場合に報告されます

状態フィールド

各状態に複数のフィールドを含めることができます。

Type	状態のタイプの説明です。たとえば、ResourcePolicyReady など。Ready 状態の場合は、他のすべての状態のサマリになります。
Status	タイプのステータスを示します。 状態には、True、False、Unknown などがあります。
Severity	Reason の分類です。 Info は、調整の実行中を意味します。 Warning は、問題が発生している可能性があるため、再試行されることを意味します。 Error は、エラーの発生により、解決のために手動による対処が必要なことを意味します。
Reason	ステータスが False である理由を示します。準備ができるまで待機していることを示す場合や、障害の理由を示す場合があります。通常は、ステータスが False の場合にスローされます。
Message	Reason について説明する、人間が解読可能な情報です。

kubectl を使用した TKG クラスタの健全性の確認

TKG コントローラがワークロード クラスタをプロビジョニングすると、いくつかのステータス状態が報告されるようになります。これらを使用することでクラスタの健全性に関する重要な情報を直接把握できます。

クラスタの健全性状態について

プロビジョニングされた TKG クラスタは、いくつかの可動部分で構成されます。これらのすべての可動部分は独立した関連性のあるコントローラによって操作され、一連の Kubernetes ノードを構築および維持するために連携します。TanzuKubernetesCluster オブジェクトと Cluster オブジェクトには、クラスタとマシンの健全性について詳細な情報を提供するステータス状態があります。

クラスタの健全性の確認

TKG クラスタの健全性を確認するには、次の手順を実行します。

1 kubectl describe cluster コマンドを実行します。

ステータスが準備完了になっている場合、クラスタ インフラストラクチャとクラスタ制御プレーンは両方とも準備ができています。例：

```
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
```

ただし、クラスタの状態が false の場合は、クラスタの準備ができておらず、メッセージ フィールドに問題の内容が示されます。たとえば、次の例では、ステータスは False であり、インフラストラクチャの準備はできていません。

```
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 クラスタの準備ができていない場合は、次のコマンドを実行して、クラスタ インフラストラクチャの問題を特定します。

```
kubectl describe vsphere cluster
```

クラスタの健全性状態のリスト

次の表に、TKG クラスタで使用可能な健全性状態の一覧と定義を示します。

条件	説明
Ready	クラスタ API オブジェクトの動作状態の概要を示します。
Deleting	基盤となるオブジェクトが現在削除されているため、ステータスは True ではありません。
DeletionFailed	基盤となるオブジェクトの削除中に問題が発生したため、ステータスは True ではありません。調整機能によって削除が再試行されるため、これは警告になります。
Deleted	基盤となるオブジェクトが削除されたため、ステータスは True ではありません。
InfrastructureReady	このクラスタに対して定義されているインフラストラクチャ オブジェクトの現在のステータスの概要を報告します。
WaitingForInfrastructure	基盤となるインフラストラクチャが利用可能になるまでクラスタが待機している場合に報告されます。注：この状態は、インフラストラクチャが準備完了状態であることを報告していない場合に、フォールバックとして使用されます。
ControlPlaneReady	クラスタ制御プレーンの準備ができている場合に報告されます。
WaitingForControlPlane	制御プレーンが利用可能になるまでクラスタが待機している場合に報告されます。注：この状態は、制御プレーンが準備完了状態であることを報告していない場合に、フォールバックとして使用されます。

状態フィールド

各状態に複数のフィールドを含めることができます。

Type	状態のタイプの説明です。たとえば、ControlPlaneReady など。Ready 状態の場合は、他のすべての状態のサマリになります。
Status	タイプのステータスを示します。 状態には、True、False、Unknown などがあります。
Severity	Reason の分類です。 Info は、調整の実行中を意味します。 Warning は、問題が発生している可能性があるため、再試行されることを意味します。 Error は、エラーの発生により、解決のために手動による対処が必要なことを意味します。
Reason	ステータスが False である理由を示します。準備ができるまで待機していることを示す場合や、障害の理由を示す場合があります。通常は、ステータスが False の場合にスローされます。
Message	Reason について説明する、人間が解読可能な情報です。

kubectl を使用した TKG クラスタのボリュームの健全性の確認

TKG クラスタにあるバインド状態のパーシステント ボリュームの健全性ステータスを確認できます。

バインド状態の各パーシステント ボリュームの健全性ステータスは、パーシステント ボリュームにバインドされているパーシステント ボリューム要求の Annotations: volumehealth.storage.kubernetes.io/messages: フィールドに表示されます。健全性ステータスには、2 つの値があります。

健全性ステータス	説明
アクセスの可能性	パーシステント ボリュームにアクセスして、使用できます。
アクセス不可	パーシステント ボリュームにアクセスしたり、使用したりできません。データストアに接続しているホストからボリュームが格納されているデータストアにアクセスできない場合、パーシステント ボリュームにはアクセスできなくなります。

手順

1 Kubectl を使用して TKG クラスタにログインします。

2 パーシステント ボリュームの要求を作成します。

a パーシステント ボリュームの要求設定を含む YAML ファイルを作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gold
  resources:
    requests:
        storage: 2Gi
```

b Kubernetes クラスタにパーシステント ボリュームの要求を適用します。

```
kubectl apply -f pvc_name.yaml
```

このコマンドでは、要求のストレージ要件を満たすバッキング仮想ディスクを持つ Kubernetes パーシステント ボリュームと vSphere ボリュームが作成されます。

c パーシステント ボリュームの要求がボリュームにバインドされているかどうかを確認します。

```
kubectl get pvc my-pvc
```

出力には、パーシステント ボリュームの要求と、ボリュームがバインド状態であることが示されます。

```
NAME     STATUS    VOLUME   CAPACITY   ACCESSMODES   STORAGECLASS   AGE
my-pvc   Bound     my-pvc   2Gi        RWO           gold           30s
```

3 ボリュームの健全性ステータスを確認します。

次のコマンドを実行して、パーシステント ボリュームにバインドされたパーシステント ボリューム要求のボリューム健全性に関する注釈を確認します。

```
kubectl describe pvc my-pvc
```

次のサンプル出力では、`volumehealth.storage.kubernetes.io/messages` フィールドに健全性ステータスがアクセス可能と表示されています。

```
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
```

Tanzu Kubernetes Grid クラスタでのボリュームの健全性の監視

バインド状態のパーシステント ボリュームの健全性ステータスを確認できます。

バインド状態の各パーシステント ボリュームの健全性ステータスは、パーシステント ボリュームにバインドされているパーシステント ボリューム要求の `Annotations: volumehealth.storage.kubernetes.io/messages:` フィールドに表示されます。健全性ステータスには、2 つの値があります。

健全性ステータス	説明
アクセスの可能性	パーシステント ボリュームにアクセスして、使用できます。
アクセス不可	パーシステント ボリュームにアクセスしたり、使用したりできません。データストアに接続しているホストからボリュームが格納されているデータストアにアクセスできない場合、パーシステント ボリュームにはアクセスできなくなります。

手順

1 vSphere Kubernetes 環境内の名前空間にアクセスします。

2 パーシステント ボリュームの要求を作成します。

a パーシステント ボリュームの要求設定を含む YAML ファイルを作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gold
  resources:
    requests:
        storage: 2Gi
```

b Kubernetes クラスタにパーシステント ボリュームの要求を適用します。

```
kubectl apply -f pvc_name.yaml
```

このコマンドでは、要求のストレージ要件を満たすバッキング仮想ディスクを持つ Kubernetes パーシステント ボリュームと vSphere ボリュームが作成されます。

c パーシステント ボリュームの要求がボリュームにバインドされているかどうかを確認します。

```
kubectl get pvc my-pvc
```

出力には、パーシステント ボリュームの要求と、ボリュームがバインド状態であることが示されます。

```
NAME     STATUS    VOLUME   CAPACITY   ACCESSMODES   STORAGECLASS   AGE
my-pvc   Bound     my-pvc   2Gi        RWO           gold           30s
```

3 ボリュームの健全性ステータスを確認します。

次のコマンドを実行して、パーシステント ボリュームにバインドされたパーシステント ボリューム要求のボリューム健全性に関する注釈を確認します。

```
kubectl describe pvc my-pvc
```

次のサンプル出力では、`volumehealth.storage.kubernetes.io/messages` フィールドに健全性ステータスがアクセス可能と表示されています。

```
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
```

vSphere Client を使用したパーシステント ボリュームの監視

DevOps エンジニアがパーシステント ボリュームの要求を使用してステートフル アプリケーションをデプロイすると、vSphere IaaS control plane により、パーシステント ボリューム オブジェクトとそれに対応するパーシステント仮想ディスクが作成されます。vSphere 管理者は、vSphere Client でパーシステント ボリュームの詳細を確認できます。また、そのストレージ コンプライアンスと健全性ステータスを監視することもできます。

手順

1 vSphere Client で、パーシステント ボリュームのある名前空間に移動します。

   a vSphere Client ホーム メニューから、[ワークロード管理] を選択します。

   b [名前空間] タブをクリックし、リストから名前空間を選択します。

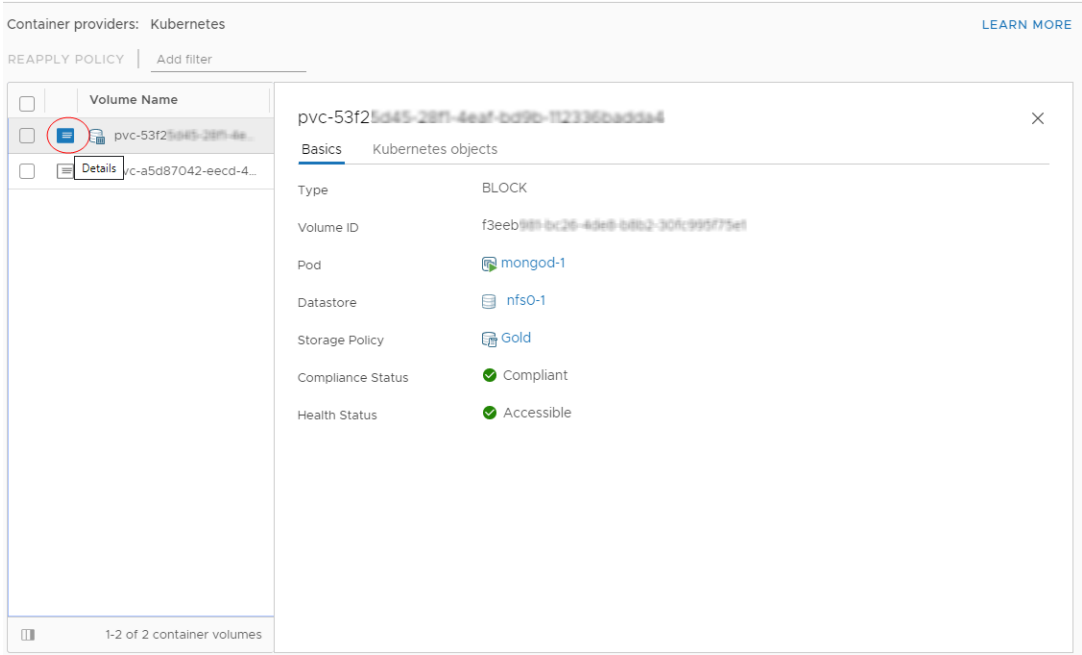
2 [ストレージ] タブをクリックし、[パーシステント ボリュームの要求] をクリックします。

   vSphere Client に、すべてのパーシステント ボリュームの要求オブジェクトと、名前空間内で使用可能な対応するボリュームが一覧表示されます。

3 選択したパーシステント ボリュームの要求の詳細を表示するには、[パーシステント ボリュームの名前] 列でボリュームの名前をクリックします。

4 [コンテナ ボリューム] 画面で、ボリュームの健全性ステータスとストレージ ポリシーのコンプライアンスを確認します。

a [詳細] アイコンをクリックし、[基本] タブと [Kubernetes オブジェクト] タブを切り替えて、Kubernetes パーシステント ボリュームの追加情報を表示します。



b ボリュームの健全性ステータスを確認します。

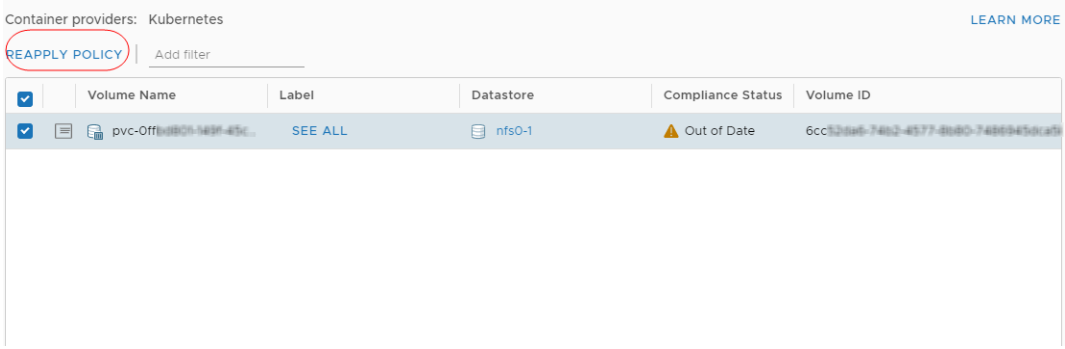
健全性ステータス	説明
アクセスの可能性	パーシステント ボリュームにアクセスして、使用できます。
アクセス不可	パーシステント ボリュームにアクセスしたり、使用したりできません。データストアに接続しているホストからボリュームが格納されているデータストアにアクセスできない場合、パーシステント ボリュームにはアクセスできなくなります。

c ストレージ コンプライアンスの状態を確認します。

[コンプライアンスの状態] 列に次のいずれかが表示されます。

コンプライアンス ステータス	説明
準拠	ボリュームをバッキングする仮想ディスクがあるデータストアには、ポリシーで必要とされるストレージ機能があります。
旧バージョン	このステータスは、ポリシーが編集されており、新しいストレージ要件がデータストアに伝送されていないことを示しています。変更を伝送するには、ポリシーを期限切れのボリュームに再適用します。
コンプライアンスに非準拠	データストアは特定のストレージ要件をサポートしますが、現在はストレージ ポリシーを満たすことができません。たとえば、データストアの物理リソースが使用不可の場合に、ステータスが「非準拠」になることがあります。ホストやディスクをクラスタに追加する方法などで、ホスト クラスタの物理構成を変更するとデータストアを準拠させることができます。その他のリソースがストレージ ポリシーを満たす場合は、ステータスが「準拠」に変わります。
該当なし	ストレージ ポリシーは、データストアでサポートされていないデータストア機能を参照しています。

d コンプライアンスの状態が「期限切れ」になっている場合は、ボリュームを選択して [ポリシーの再適用] をクリックします。



ステータスが「準拠」になります。

Kubectl を使用した TKG クラスタ シークレットの取得

TKG クラスタはシークレットを使用して、運用するトークン、キー、およびパスワードを格納します。

TKG クラスタ シークレットのリスト

Kubernetes シークレットは、パスワード、トークン、SSH キーなどの少量の秘密データを格納するオブジェクトです。TKG クラスタの管理者は、クラスタの運用中に複数のシークレットを使用することがあります。この表には、クラスタ管理者が使用する可能性のある主なシークレットとその説明が示されています。

注： このリストは、完全なものではありません。このリストには、手動でのローテーションが必要になるシークレットや、トラブルシューティングのためにクラスタ ノードにアクセスする際に使用する必要があるシークレットのみが含まれています。

シークレット	説明
TANZU-KUBERNETES-CLUSTER-NAME-ccm-token-RANDOM	vSphere 名前空間 に接続するために準仮想化クラウド プロバイダのクラウド コントローラ マネージャによって使用されるサービス アカウント トークン。この認証情報のローテーションをトリガするには、シークレットを削除します。
TANZU-KUBERNETES-CLUSTER-NAME-pvcsi-token-RANDOM	vSphere 名前空間 に接続するために準仮想化 CSI プラグインによって使用されるサービス アカウント トークン。この認証情報のローテーションをトリガするには、シークレットを削除します。
TANZU-KUBERNETES-CLUSTER-NAME-kubeconfig	`kubernetes-admin` ユーザーとしてクラスタの制御プレーンに接続する際に使用可能な kubeconfig ファイル。vCenter Single Sign-On 認証が使用できない場合に、このシークレットを使用してクラスタにアクセスして、トラブルシューティングを行うことができます。
TANZU-KUBERNETES-CLUSTER-NAME-ssh	`vmware-system-user` として任意のクラスタ ノードに接続する際に使用可能な SSH プライベート キー。このシークレットを使用して、任意のクラスタ ノードに SSH 接続し、トラブルシューティングを行うことができます。
TANZU-KUBERNETES-CLUSTER-NAME-ssh-password	`vmware-system-user` として任意のクラスタ ノードに接続する際に使用可能なパスワード。
TANZU-KUBERNETES-CLUSTER-NAME-ca	`kubectl` で Kubernetes API サーバに安全に接続するために使用する Tanzu Kubernetes クラスタ制御プレーンのルート CA 証明書。

Kubectl を使用した TKG クラスタ ネットワークの確認

システムでは、ノード、ポッド、サービスのためのデフォルトのネットワークを使用して、TKG クラスタをプロビジョニングします。カスタム kubectl コマンドを使用してクラスタ ネットワークを確認できます。

TKG クラスタ ネットワークを確認するためのカスタム コマンド

クラスタ ネットワークの検証には、次のコマンドを参照してください。

これらのコマンドは、TKG クラスタがプロビジョニングされている vSphere 名前空間 から実行する必要があります。例：

```
kubectl config use-context tkg2-cluster-ns
```

表 8-10. クラスタ ネットワークを確認するためのカスタム kubectl コマンド

コマンド	説明

コマンド
`kubectl get tkgserviceconfigurations`
結果の例
`NAME DEFAULT CNI`
`tkg-service-configuration antrea`
	デフォルトの CNI（変更されていなければ `antrea`）を返します。
	クラスタの YAML で明示的にオーバーライドされていないかぎり、クラスタの作成にはデフォルトの CNI が使用されます。

コマンド
`kubectl get virtualnetwork`
結果の例
`NAME SNAT`
`READY AGE`
`tkgs-cluster-12-vnet 10.191.152.133`
`True 4h3m`
	クラスタ ノードの仮想ネットワークを返します。
	ネットワーク アドレス変換 (SNAT) IP アドレスが割り当てられていることを確認するために使用します。

コマンド
`kubectl get virtualmachines -o wide`
結果の例
`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`
	クラスタ ノードの仮想ネットワーク インターフェイスを返します。
	各クラスタ ノードの仮想マシンに IP アドレスが割り当てられていることを確認するために使用します。

コマンド
`kubectl get virtualmachineservices`
結果の例
`NAME`
`TYPE AGE`
`tkg2-cluster-12-control-plane-service`
`LoadBalancer 3h53m`
	各クラスタ ノードの仮想マシン サービスを返します。
	状態が更新されており、ロード バランサの仮想 IP (VIP) アドレスが含まれていることを確認するために使用します。

表 8-10. クラスタ ネットワークを確認するためのカスタム kubectl コマンド （続き）

コマンド	説明

コマンド
`kubectl get services -n NAMESPACE`
cURL を使用した確認
`curl -k https://EXTERNAL-IP:PORT/healthz`
	クラスタ API のアクセス用に作成された Kubernetes サービス ロード バランサを返します。外部 IP アドレスが割り当てられていることを確認するために使用します。
	ロード バランサ サービスの外部 IP アドレスおよびポートを使用した API へのアクセスを確認するには、`curl` を使用します。

コマンド
`kubectl get endpoints`
結果の例
`NAME ENDPOINTS AGE tkg2-cluster-12-control-plane-service 10.244.0.66:6443 3h44m`
	クラスタの制御プレーン ノード（エンドポイント）を返します。各エンドポイントが作成され、エンドポイント プールに含まれていることを確認するために使用します。

kubectl を使用した TKG クラスタの操作の確認

カスタム kubectl コマンドを使用して、TKG クラスタを管理できます。これらのコマンドは、TKG コントローラによって管理されるカスタム リソースによって提供されます。

TKG クラスタを管理するためのカスタム コマンド

次の表に、TKG クラスタを管理するための kubectl コマンドの一覧と説明を示します。

各コマンドは、TKG クラスタがプロビジョニングされている vSphere 名前空間 のコンテキストから実行します。クラスタ コンテキストでこれらのコマンドを実行しても情報は返されません。

表 8-11. TKG クラスタを管理するためのカスタム コマンド

コマンド	説明
`kubectl get tanzukubernetescluster`	現在の名前空間内にある TKC のリストを表示します。
`kubectl get tkc`	前のコマンドの短縮形バージョンです。
`kubectl get cluster`	名前空間内にあるクラスタを返します。
`kubectl describe tanzukubernetescluster CLUSTER-NAME`	指定されたクラスタを説明し、示されている状態、ステータス、およびイベントを表示します。プロビジョニングが完了すると、このコマンドは、Kubernetes API エンドポイントに対応するロード バランサ用に作成された仮想 IP アドレスを示します。

表 8-11. TKG クラスタを管理するためのカスタム コマンド （続き）

コマンド	説明
`kubectl get cluster-api`	現在の名前空間でクラスタをサポートしているクラスタ API リソースのリストを表示します。これには、クラスタ API プロジェクトのリソース、および Tanzu Kubernetes Grid サービスによって使用されるクラスタ API プロバイダのリソースも含まれます。
`kubectl get tanzukubernetesreleases`	利用可能な Tanzu Kubernetes リリースのリストを表示します。
`kubectl get tkr`	前のコマンドの短縮形バージョンです。
`kubectl get tkr v1.23.8---vmware.1-tkg.1.5417466 -o yaml`	指定した Tanzu Kubernetes リリースの詳細を提供します。
`kubectl get virtualmachine`	現在の名前空間内のクラスタ ノードをサポートしている仮想マシン リソースのリストを表示します。
`kubectl get vm`	前のコマンドの短縮形バージョンです。
`kubectl describe virtualmachine VIRTUAL-MACHINE-NAME`	指定した仮想マシンを説明し、状態、現在のステータス、およびイベントを表示します。
`kubectl describe virtualmachinesetresourcepolicy`	現在の名前空間内のクラスタをサポートしている仮想マシン セット リソース ポリシー リソースのリストを表示します。このリソースは、クラスタに使用される vSphere オブジェクトのリソース プールとフォルダを表します。
`kubectl get virtualmachineservice`	現在の名前空間内のクラスタ ノードをサポートしている仮想マシン サービス リソースのリストを表示します。これらのリソースは、サービスに似ていますが、ポッドではなく仮想マシン用です。仮想マシン サービスは、クラスタの制御プレーン ノードのロード バランサを提供するために使用され、クラスタ内で LoadBalancer タイプの Kubernetes サービスをサポートするために準仮想化クラウド プロバイダにも使用されます。
`kubectl get vmservice`	前のコマンドの短縮形バージョンです。
`kubectl describe virtualmachineservice VIRTUAL-MACHINE-SERVICE-NAME`	指定した仮想マシン サービスを説明し、クラスタの示された状態、現在のステータス、およびイベントを表示します。
`kubectl get virtualmachineimage`	利用可能な仮想マシン イメージのリストを表示します。
`kubectl get vmimage`	前のコマンドのショートカット バージョンです。
`kubectl describe vmimage VM_IMAGE_NAME`	名前付き仮想マシン イメージの詳細情報を表示します。
`kubectl get virtualnetwork`	現在の名前空間内の仮想ネットワーク リソース（クラスタに使用されているものも含む）のリストを表示します。仮想ネットワークは、クラスタがプロビジョニングされる名前空間ごと、および各クラスタに対して作成されます。

表 8-11. TKG クラスタを管理するためのカスタム コマンド （続き）

コマンド	説明
`kubectl get persistentvolumeclaim`	現在の名前空間内のパーシステント ボリューム要求リソース（クラスタに使用されているものも含む）のリストを表示します。
`kubectl get cnsnodevmattachment`	現在の名前空間内の CNS ノード仮想マシン接続リソースのリストを表示します。これらのリソースは、クラスタのノードとして機能する仮想マシンへの、CNS によって管理されるパーシステント ボリュームの接続を表します。
`kubectl get configmap`	現在の名前空間内の構成マップ（クラスタ ノードの作成に使用されるものも含む）のリストを表示します。構成マップはユーザーが変更できるものではなく、すべての変更が上書きされます。
`kubectl get secret`	現在の名前空間内のシークレット（クラスタ ノードの作成および管理に使用されるシークレットも含む）のリストを表示します。

TKG クラスタのライフサイクル ステータスの表示

kubectl を使用して、vSphere インベントリ内の TKG クラスタのライフサイクル ステータスを表示できます。

vSphere での TKG クラスタのライフサイクル ステータス

以下の表に、vSphere インベントリに表示される TKG クラスタのステータス情報を示します。

表 8-12. vSphere インベントリ内の TKG クラスタのステータス

フィールド	説明	例
名前	クラスタのユーザー定義名。	`tkg2-cluster-01`
作成時間	クラスタの作成日時。	`Mar 17, 2022, 11:42:46 PM`
段階	クラスタのライフサイクル ステータス。	`creating`
ワーカー数	クラスタ内のワーカー ノードの台数。	`1`、`2`、または `5`
ディストリビューション バージョン	クラスタが実行している Kubernetes ソフトウェアのバージョン。	`v1.22.6+vmware.1-tkg.1.7144628`
制御プレーンのアドレス	クラスタ制御プレーンのロード バランサの IP アドレス。	`192.168.123.2`

kubectl での TKG クラスタのライフサイクル ステータス

この表には、kubectl に表示される TKG クラスタのステータス情報が示されています。

表 8-13. kubectl での TKG クラスタのステータス

フィールド	説明	例
NAME	クラスタの名前。	`tkg2-cluster-01`
CONTROL PLANE	クラスタ内の制御プレーン ノードの台数。	`3`

表 8-13. kubectl での TKG クラスタのステータス （続き）

フィールド	説明	例
WORKER	クラスタ内のワーカー ノードの台数。	`5`
DISTRIBUTION	クラスタが実行されている Kubernetes バージョン。	`v1.22.6+vmware.1-tkg.1.5b5608b`
AGE	クラスタが実行されている日数。	`13d`
PHASE	クラスタのライフサイクル ステータス。	`running`

クラスタのライフサイクル フェーズのステータス

この表には、クラスタ ライフサイクルの各フェーズのステータスが示されています。

表 8-14. クラスタのライフサイクル フェーズのステータス

段階	説明
`creating`	クラスタのプロビジョニングを開始可能か、制御プレーンが作成中か、制御プレーンが作成中で未初期化のいずれかの段階です。
`deleting`	クラスタの削除中です。
`failed`	クラスタ制御プレーンの作成に失敗し、ユーザーの操作が必要な場合があります。
`running`	インフラストラクチャが作成および構成されていて、制御プレーンが完全に初期化されています。
`updating`	クラスタの更新中です。

kubectl を使用した TKG クラスタのリソース階層の表示

kubectl を使用して、TKG クラスタのリソース階層を表示できます。クラスタ リソースの完全なリストを表示すると、問題の原因となっている可能性のあるリソースを特定するのに役立ちます。

手順

1 スーパーバイザー に接続してログインします。

2 TKG クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context tkg2-cluster-ns
```

3 次のコマンドを実行して、クラスタ API クラスタ リソースを表示します。

```
kubectl describe clusters.cluster.x-k8s.io CLUSTER-NAME
```

このコマンドは、Kubernetes 名前空間、API のバージョン、リソースのバージョンなど、指定したクラスタのリソース階層を返します。

v1beta1 クラスタの MachineHealthCheck の構成

このトピックでは、v1beta1 API を使用してプロビジョニングされた TKG サービス クラスタの MachineHealthCheck を構成する方法について説明します。

v1beta1 クラスタの MachineHealthCheck

MachineHealthCheck は、健全でないマシンを修正するための条件を定義する Kubernetes クラスタ API リソースです。Kubernetes では、マシンは kubelet を実行できるカスタム リソースです。vSphere IaaS control plane では、Kubernetes マシン リソースが vSphere 仮想マシンによってバッキングされます。詳細については、アップストリームのドキュメントを参照してください。

TKG サービス を使用してクラスタをプロビジョニングすると、デフォルトの MachineHealthCheck オブジェクトが作成されます。1 つはすべての制御プレーン用、もう 1 つは各マシンのデプロイ用です。vSphere 8 Update 3 以降では、v1beta1 クラスタに対してマシンの健全性チェックを構成できます。サポートされる設定は次のとおりです。

- maxUnhealthy
- nodeStartupTimeout
- unhealthyConditions
- unhealthyRange

次の表に、サポートされているマシンの健全性チェックの操作を示します。

表 8-15. マシンの健全性チェック

フィールド	値	説明
`maxUnhealthy`	文字列 絶対数またはパーセンテージ	健全でないマシンの数がこの値を超えると、修正は実行されません。
`nodeStartupTimeout`	文字列 `XhXmXs` 形式（時間、分、秒）の期間	クラスタに参加する期間よりも作成に時間がかかるマシンは、問題があると見なされて修正されます。

表 8-15. マシンの健全性チェック （続き）

フィールド	値	説明
unhealthyConditions	unhealthyConditions タイプの配列 [] 使用可能な条件タイプは次のとおりです。 [Ready, MemoryPressure, DiskPressure, PIDPressure, NetworkUnavailable] 使用可能な条件ステータスは次のとおりです。 [True, False, Unknown]	制御プレーン ノードが健全でないと見なされるかどうかを決定する条件のリスト。
unhealthyRange	文字列	「セレクタ」によって健全でないとして選択されたマシンの数が unhealthyRange の範囲内にある場合にのみ追加の修正が許可されます。 maxUnhealthy よりも優先されます。たとえば、「[3-5]」は、(a) 3 台以上の健全でないマシンがあり、（かつ） (b) 最大 5 台の健全でないマシンがある場合にのみ修正が許可されることを意味します。

注： MachineHealthCheck オブジェクトは v1alpha3 クラスタ用にデプロイされますが、構成できません。詳細については、kubectl を使用した TKG クラスタのマシンの健全性の確認を参照してください。

MachineHealthCheck の例

次の例では、特定の machineDeployment の machineHealthCheck を構成します。

```
...
  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
```

Kubectl を使用した MachineHealthCheck のパッチ適用

v1beta1 クラスタのプロビジョニング後にそのクラスタの `MachineHealthCheck` を更新するには、`patch` メソッドを使用します。

注意： 以下の手順では、既存のクラスタへのパッチ適用に関する一般的なガイダンスを示します。使用する値は、ご使用の環境とパッチを適用するデプロイ済みのクラスタによって異なります。Tanzu CLI を使用して、既存のクラスタの `MachineHealthCheck` にパッチを適用することを検討してください。

1 クラスタ リソース定義から `machineDeployment` を取得します。

```
kubectl get cluster CLUSTER_NAME -o yaml
```

`spec.topology.workers.machineDeployments` セクションには、各 `machineDeployment` を識別する値が表示されます。

2 ワーカー ノードの MachineHealthCheck を削除します。

```
kubectl patch cluster <Cluster Name> -n <cluster namespace>  --type json -p='{"op":
"replace", "path": "/spec/topology/workers/machineDeployments/<index>/machineHealthCheck",
"value":{"enable":false}}'
```

3 制御プレーンの MachineHealthCheck を削除します。

```
kubectl patch cluster <cluster-name> -n <cluster-namespace> --type json -p='{"op":
"replace", "path": "/spec/topology/controlPlane/machineHealthCheck", "value":
{"enable":false}}'
```

4 目的の設定を使用して、制御プレーンの MachineHealthCheck を作成または更新します。

```
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 目的の設定を使用して、ワーカー ノードの MachineHealthCheck を作成または更新します。

```
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%"}}]'
```

Tanzu CLI を使用した MachineHealthCheck の構成

Tanzu CLI を使用して、v1beta1 クラスタの MachineHealthCheck を構成できます。

たとえば、制御プレーンの MachineHealthCheck の設定を作成または更新するには、次のコマンドを実行します。

```
tanzu cluster mhc control-plane set <cluster-name> --node-startup-timeout 2h7m10s
```

設定が更新されたかどうか、および調整されていないかどうかを確認するには、次のコマンドを実行します。

```
tanzu cluster mhc control-plane get <cluster-name>
```

マシンのデプロイの MachineHealthCheck の設定を作成または更新するには、次のコマンドを実行します。

```
tanzu cluster mhc node set <cluster-name> --machine-deployment node-pool-1 --node-startup-
timeout 1h59m0s
```

設定が更新されたかどうか、および調整されていないかどうかを確認するには、次のコマンドを実行します。

```
tanzu cluster mhc node get <cluster-name> -m <cluster-name>-node-pool-1-nr7r5
```

get と set に加えて、システムでは削除 (delete) 操作がサポートされます。例：

制御プレーンでは、次のコマンドを使用できます。

```
tanzu cluster mhc control-plane delete <cluster-name>
```

ノードでは、次のコマンドを使用できます。

```
tanzu cluster mhc <cluster-name> --machine-deployment <machine deployment name>
```

TKG サービス クラスタの更新

9

このセクションでは、TKG サービス クラスタの更新について説明します。

次のトピックを参照してください。

- TKG サービス クラスタのローリング アップデート モデルについて
- 更新のための TKGS クラスタ互換性の確認
- TKR バージョンの編集による TKG クラスタの更新
- ストレージ クラスの編集による TKG クラスタの更新
- 仮想マシン クラスの編集による TKG サービス クラスタの更新
- Tanzu CLI を使用した TKG クラスタの更新

TKG サービス クラスタのローリング アップデート モデルについて

TKG サービス クラスタでは、ローリング アップデート モデルがサポートされます。クラスタ仕様を変更すると、ローリング アップデートを開始できます。一部のシステム操作でローリング アップデートが開始される場合があります。環境をアップデートする前に、ローリング アップデート プロセスについて理解しておく必要があります。

TKG サービス 3.0 以降の TKGS クラスタのローリング アップデート モデル

TKG サービス 3.0 以降では、TKG コントローラが vCenter Server とスーパーバイザーから独立しています。TKG サービス の使用を参照してください。これらのコンポーネントをアップグレードしても、TKGS クラスタのローリング アップデートは開始されません。

TKG サービス バージョンをアップグレードすると、TKGS クラスタのローリング アップデートがトリガされる場合があります。

TKG サービス 3.0 より前の TKGS クラスタのローリング アップデート モデル

TKG コントローラは、スーパーバイザー で実行されます。スーパーバイザー をアップデートすると、アップデートが可能な場合は TKG コントローラが自動的にアップデートされます。各 TKG コントローラのアップデートには、CNI、CSI、CPI などのサポート サービスのアップデートと、クラスタの構成のアップデートが含まれる可能性があります。互換性を維持するために、システムは事前チェックを実行し、コンプライアンスを適用します。

vSphere IaaS control plane では、ローリング アップデート モデルを使用して、スーパーバイザー 上の TKG クラスタをアップデートします。ローリング アップデート モデルを使用すると、クラスタをアップデートしている間のダウンタイムを最小限に抑えられます。ローリング アップデートには、Kubernetes ソフトウェアのバージョンのアップグレードに加えて、仮想マシンの構成とリソース、サービスと名前空間、カスタム リソースなどの、クラスタをサポートするインフラストラクチャおよびサービスの更新が含まれます。アップデートが正常に実行されるためには、構成がいくつかの互換性要件を満たしている必要があります。そのため、システムは再チェック条件を適用して、クラスタの更新の準備ができていることを確認し、クラスタのアップグレードが失敗した場合のロールバックをサポートします。

クラスタ マニフェストの特定の要素を変更することで、TKG クラスタのローリング アップデートを開始できます。ローリング クラスタ アップデートは、システムによって開始することもできます。たとえば、vSphere 名前空間の更新を実行すると、システムは更新された構成を直ちにすべてのワークロード クラスタに伝達します。これらの更新によってクラスタ ノードのローリング アップデートをトリガすることができます。また、いずれかの構成要素に対する変更によってローリング アップデートを開始することもできます。たとえば、ディストリビューション バージョンに対応する `VirtualMachineImage` を名前変更または置換すると、システムが新しいイメージで実行されているすべてのノードの取得を試行するため、ローリング アップデートが開始されます。スーパーバイザー を更新した場合も、そこにデプロイされているワークロード クラスタのローリング アップデートがトリガされることがあります。たとえば、`vmware-system-tkg-controller-manager` が更新された場合、システムは新しい値をマニフェスト ジェネレータに導入し、コントローラはこれらの値をデプロイするローリング アップデートを開始します。

クラスタ ノードを置き換えるためのローリング アップデート プロセスは、Kubernetes 環境でのポッドのローリング アップデートと同様です。ワークロード クラスタのローリング アップデートを実行するコントローラは 2 つあります。アドオン コントローラとクラスタ コントローラです。これらの 2 つのコントローラでは、ローリング アップデートに、アドオンの更新、制御プレーンの更新、およびワーカー ノードの更新の 3 つの主要なステージがあります。これらのステージは順番に実行されますが、前の手順が十分に進行するまで次のステップの開始を防ぐ事前チェックが実行されます。不要と判断された場合、ステップはスキップされることがあります。たとえば、更新がワーカー ノードのみに影響する場合、アドオンや制御プレーンの更新は必要なくなります。

更新プロセスでは、システムは新しいクラスタ ノードを追加し、ノードがターゲットの Kubernetes バージョンでオンラインになるまで待機します。その後、古いノードを削除対象としてマークし、次のノードに移動して、プロセスを繰り返します。すべてのポッドが削除されるまで、古いノードは削除されません。たとえば、ノードの完全なドレーンを妨げる PodDisruptionBudgets でポッドが定義されている場合、ノードは遮断されますが、それらのポッドが消去できるようになるまで削除されません。システムは、最初にすべての制御プレーン ノード、次にワーカー ノードをアップグレードします。更新中は、クラスタのステータスが「更新中」に変わります。ローリング アップデート プロセスが完了すると、クラスタのステータスが「実行中」に変わります。

レプリケーション コントローラによって管理されていないクラスタで実行されているポッドは、クラスタの更新中にワーカー ノードのドレインの一環として、Kubernetes バージョンのアップグレードで削除されます。クラスタの更新が手動でトリガされた場合や、vSphere 名前空間または スーパーバイザー の更新によって自動実行された場合がこれに該当します。レプリケーション コントローラによって管理されていないポッドには、Deployment または ReplicaSet 仕様の一部として作成されていないポッドなどがあります。詳細については、Kubernetes ドキュメントの Pod Lifecycle: Pod lifetime を参照してください。

ユーザーが開始するローリング アップデート

スーパーバイザー で TKG クラスタのローリング アップデートを開始するには、Tanzu Kubernetes リリース バージョン、仮想マシン クラス、ストレージ クラスをそれぞれアップデートします。詳細については、次のいずれかのトピックを参照してください。

- TKR バージョンの編集による TKG クラスタの更新
- ストレージ クラスの編集による TKG クラスタの更新
- 仮想マシン クラスの編集による TKG サービス クラスタの更新
- Tanzu CLI を使用した TKG クラスタの更新

システムが開始するローリング アップデート

スーパーバイザー のリリースごとに、次の 1 つ以上のオブジェクトを変更できます。

- kubeadmcontrolplanetemplate/kubeadmcontrolplane
- kubeadmconfigtemplate/kubeadmconfig
- vspheremachinetemplate/vspheremachine（vSphere 8.x の場合）
- wcpmachinetemplate/wcpmachine（vSphere 7.x の場合）

スーパーバイザー がアップグレードされると、コア クラスタ API (CAPI) コントローラは TKG ワークロード クラスタへのアップデート ロールアウトをトリガし、上記のオブジェクトの目的の状態を、実行中のワークロード クラスタと一致させます。

vSphere IaaS control plane では、スーパーバイザー で実行されている TKG コントローラによってこれらのオブジェクトが生成され、システム コードとの同期が維持されます。これにより、コントローラが新しいコードに更新された場合、上記のオブジェクトのいずれかを変更すると、既存の TKG クラスタがローリング アップデートされます。つまり、スーパーバイザー に影響する変更をシステム コードに加えると、TKG クラスタがローリング アップデートされます。

次の表に、スーパーバイザー がアップグレードされたときにワークロード クラスタのローリング アップデートが自動実行される条件を示します。

アップグレードのシナリオ	説明
任意の vCenter Server 7.x リリースから任意の vCenter Server リリースへのアップグレード	すべての Tanzu Kubernetes クラスタのローリング アップデートがトリガされる場合があります。 ローリング アップデートは、vCenter Server のアップグレードに続くスーパーバイザーの最初のアップグレードによってトリガされます。通常、ローリング アップデートは、同じ vCenter Server でのスーパーバイザーのアップグレードによってトリガされることはありません。 詳細については、リリース ノートを参照してください。
任意の vCenter Server リリースから任意の vCenter Server 8.x リリースへのアップグレード	次のコード変更を反映させる必要があるため、すべての TKG クラスタのローリング アップデートがトリガされます。 ■ 基盤となる CAPI プロバイダを CAPW から CAPV に移動する ■ クラスタをクラスレス CAPI クラスタから上位の CAPI クラスタに移行する

アップグレードのシナリオ	説明
vCenter Server 8.0 GA リリース (8.0.0) から vCenter Server 8.0.0b または 8.0.0c リリースへのアップグレード	次のいずれかのケースが当てはまる場合は、指定した TKG クラスタのローリング アップデートがトリガされます。 ■ 空でない noProxy リストを持つプロキシ設定が使用されていた TKG クラスタがある場合は、この TKG クラスタ。 ■ スーパーバイザーで組み込みの Habor レジストリ サービスが有効になっていた場合は、すべての TKG クラスタ。
vSphere 8.0.0b リリースから vSphere 8.0.0c リリースへのアップグレード	ワークロード クラスタの自動ロールアウトなし
vSphere 8.0.0c リリースから vSphere 8.0 Update 1 リリース (8.0.1) へのアップグレード	ワークロード クラスタの自動ロールアウトなし
任意の vSphere 8.x バージョンから 8.0 U2 リリース (8.0.2) へのアップグレード	以下の変更を行う必要があるため、すべての TKC に対するローリング アップグレードが実行されます。 ■ vSphere 8.0 U2 には、ClusterClass の一部として、GCM 内の TKG 1.0 と TKG 2.0 の両方の TKR に対する Kubernetes レベルの STIG の変更が含まれています。 ■ 1.23 以降の TKC には 8.0 U2 に対する互換性があるため、すべてのクラスタでローリング アップグレードが実行されます。
8.0 U2 (8.0.2) 未満の任意の vSphere 8.x バージョンから 8.0 U2c リリースへのアップグレード	以下の変更を行う必要があるため、すべての TKC に対するローリング アップグレードが実行されます。 ■ 8.0 U2 には、ClusterClass の一部として、GCM 内の TKG 1.0 と TKG 2.0 の両方の TKR に対する k8s レベルの STIG の変更が含まれています。 ■ 1.23 以降の TKC には 8.0 P03 に対する互換性があるため、すべてのクラスタでローリング アップグレードが実行されます。

また、TKR イメージをホストしているコンテンツ ライブラリを変更したときに、TKG クラスタのローリング アップデートがトリガされることがあります。サブスクリプションを介して、または手動で新しいイメージを追加しても、TKG クラスタのローリング アップデートはトリガされません。ただし、コンテンツ ライブラリを変更して、異なる名前のイメージを追加すると、すべての TKG クラスタのローリング アップデートがトリガされます。

たとえば、システム定義の OVA 名を自動的に使用するサブスクライブ済みコンテンツ ライブラリを使用しているシナリオについて考えます。ここでは、ローカル コンテンツ ライブラリに切り替え、同じ OVA をポピュレートしてそれぞれ異なる名前を指定します。これにより、すべての TKG クラスタのローリング アップデートがトリガされます。置換後のコンテンツ ライブラリには同じ OVA がありますが、ユーザー定義の異なる名前を持つためです。

複数のノード プールを持つクラスタのローリング アップデートに関する考慮事項

複数のノード プールを持つ TKG クラスタを使用している場合は、ローリング アップデートに関する次の情報を考慮してください。

ワーカー ノード プール

ワーカー ノード プールは、vSphere 7 U3 でリリースされた TKGS v1alpha2 API で導入されました。クラスタ API MachineDeployments は、ワーカー ノード プールの基盤となる Kubernetes プリミティブです。

ClusterClass は、vSphere 8 リリースの TKGS で導入されました。v1alpha3 API と v1beta1 API の両方が ClusterClass に基づいています（v1alpha3 は ClusterClass 上の抽象化レイヤーです）。

ローリング アップデート中に複数のノード プールを更新する方法

複数のノード プールを使用してプロビジョニングされた TKGS ワークロード クラスタを更新する場合は、使用されている vSphere のバージョンによってローリング アップデート モデルが異なります。

vSphere	TKGS API	アップグレードの動作
vSphere 7 TKGS	v1alpha2 API	同じクラスタ内の複数のノード プールが同時に更新されます
vSphere 8 TKGS	v1alpha3 API および v1beta1 API	同じクラスタ内の複数のノード プールが論理的な順序に従って（順番に）更新されます

ベスト プラクティスに関する考慮事項

同一の複数のノード プールを使用して vSphere 8 TKGS クラスタをプロビジョニングしても、サイジングの観点からは意味がありません。ノード プールは、サイズ、仮想マシン クラス、TKr バージョンなどが異なる場合に使用する必要があります。複数の同一のノード プールを使用してクラスタを迅速にアップグレードしても効果がないため、この処理は行わないでください。

Pod Disruption Budget は、アップグレードが実行中のアプリケーションに干渉しないようにするための適切な方法です。これに対処する最善の方法は、ワークロードに PodDisruptionBudgets を設定することです（https://kubernetes.io/docs/tasks/run-application/configure-pdb/ を参照）。クラスタ API はこれらの設定を考慮し、しきい値を超えるとマシンを終了しません。

vSphere 8 TKGS クラスタのローリング アップデートの詳細

vSphere 8 TKGS クラスタ バージョンの更新中：

- 制御プレーン ノードが最初に更新され、続いて Zone-A ノード プールから一度に 1 つのワーカー ノードがロールアウトされます。2 つのノード プールが使用されている場合、一度にロールアウトされるワーカーは 1 つのみです。

クラスタ構成変数の更新中：

- 制御プレーン ノードが最初に更新され、続いてノード プールごとに 1 つのワーカー ノードがロールアウトされます。たとえば、2 つのノード プールが使用されている場合は、一度に 2 つのワーカーがロールアウトされます。

更新のための TKGS クラスタ互換性の確認

TKGS ワークロード クラスタをアップグレードする前に、クラスタがアップグレードに対する互換性を備えているか確認する必要があります。TKG サービス に対する互換性を再確認する必要があります。

TKG サービス による互換性の確認

ワークロード クラスタをアップグレードする前に、クラスタがアップグレードに対する互換性を備えているか確認する必要があります。クラスタに TKG サービス との互換性がない場合は、Tanzu Kubernetes リリース をアップグレードします。使用可能な TKr の詳細については、リリース ノードを参照してください。オンライン相互運用性マトリックスも参照してください。

次のコマンドを使用して、Tanzu Kubernetes リリース の一覧および互換性を表示できます。

```
kubectl get tkr
```

COMPATIBLE 列は、その Tanzu Kubernetes リリース がインストール済みの TKG サービス と互換性があるかどうかを示します。TKG サービス 3.1 リリース以降では、TYPE 列も互換性ステータスを返します。

TKGS クラスタを指定した場合は、使用可能な TKr のアップデートが表示されます。

v1alpha3 API を使用する場合：

```
kubectl get tkc <tkgs-cluster-name>
```

または v1beta1 API を使用する場合：

```
kubectl get cc <tkgs-cluster-name>
```

UPDATES AVAILABLE 列は、Kubernetes の利用可能なアップグレードがあるかどうかと、使用する次の推奨 Tanzu Kubernetes リリース を示します。例：

```
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]
```

TKr 形式には、非レガシーとレガシーの 2 種類があります。

- 非レガシー TKr は vSphere 8.x 専用であり、vSphere 8.x とのみ互換性があります
- レガシー TKr は vSphere 7.x および vSphere 8.x と互換性があるレガシー形式を使用していますが、アップグレードのみを目的としています。

非レガシー TKr を一覧表示する場合：

```
kubectl get -l !run.tanzu.vmware.com/legacy-tkr
```

レガシー TKr を一覧表示する場合：

```
kubectl get -l !run.tanzu.vmware.com/legacy-tkr
```

TKR バージョンの編集による TKG クラスタの更新

このタスクでは、TKG クラスタ マニフェストを編集して TKG クラスタの Tanzu Kubernetes リリース バージョンを更新する方法について説明します。

`kubectl edit` コマンドを使用して Tanzu Kubernetes リリース バージョンをアップグレードすることで、TKGS クラスタのローリング アップデートを開始できます。

注： `kubectl apply` コマンドを使用して、デプロイされたクラスタの TKR バージョンを更新することはできません。

前提条件

このタスクでは、kubectl edit コマンドを使用する必要があります。このコマンドを実行すると、KUBE_EDITOR または EDITOR 環境変数によって定義されたテキスト エディタで、クラスタ マニフェストが開かれます。ファイルを保存すると、変更が反映されてクラスタが更新されます。`kubectl edit` コマンドを実行できるように kubectl のエディタを構成するには、「kubectl のテキスト エディタの構成」を参照してください。

手順

1 スーパーバイザー で認証します。

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 ターゲット ワークロード クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 ターゲット TKG クラスタおよびバージョンを取得します。

v1alpha3 クラスタ：

```
kubectl get tanzukubernetescluster
```

v1beta1 クラスタ：

```
kubectl get cluster
```

4 使用できる Tanzu Kubernetes リリースをリストします。

```
kubectl get tanzukubernetesreleases
```

5 次のコマンドを実行して、クラスタのマニフェストを編集します。

v1alpha3 クラスタ：

```
kubectl edit tanzukubernetescluster/CLUSTER-NAME
```

v1beta1 クラスタ：

```
kubectl edit cluster/CLUSTER-NAME
```

6 Tanzu Kubernetes リリース の文字列を更新してマニフェストを編集します。

たとえば、v1alpha3 クラスタの場合、TKR v1.25.7 を次のように変更します。

```
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
```

TKR v1.26.5 に変更：

```
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
```

注： 制御プレーン ノードとワーカー ノードの TKR バージョンが同じである必要があります。すべての TKr インスタンスを更新するか、制御プレーンのバージョンを更新して、ワーカー ノードから TKR 名を削除できます。

たとえば、v1beta1 クラスタの場合、TKR v1.25.7 を次のように変更します。

```
apiVersion: cluster.x-k8s.io/v1beta1
  ...
  topology:
    class: tanzukubernetescluster
    version: v1.25.7---vmware.3-fips.1-tkg.1
    controlPlane:
      replicas: 3
    workers:
      ...
    variables:
      ...
```

TKR v1.26.5 に変更：

```
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 マニフェスト ファイルに行った変更内容を保存します。

ファイルを保存すると、kubectl によって変更がクラスタに適用されます。バックグラウンドで、スーパーバイザーの仮想マシン サービスによって新しいワーカー ノードがプロビジョニングされます。

8 kubectl から、マニフェストの編集が正常に記録されたことが報告されているかを確認します。

```
kubectl edit tanzukubernetescluster/tkg-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkg-cluster-1 edited
```

注： エラーが表示された場合、またはクラスタ マニフェストが正常に編集されたことが kubectl から報告されない場合は、KUBE_EDITOR 環境変数を使用して、デフォルトのテキスト エディタが適切に構成されていることを確認してください。kubectl のテキスト エディタの構成を参照してください。

9 クラスタが更新されていることを確認します。

```
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 クラスタが更新されたことを確認します。

```
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
```

ストレージ クラスの編集による TKG クラスタの更新

クラスタ ノードで使用されるストレージ クラスを変更して、TKG クラスタを更新できます。

`kubectl edit` コマンドを使用してクラスタ仕様の `storageClass` パラメータの値を編集することで、TKG クラスタのローリング アップデートを開始できます。

注： `kubectl apply` コマンドを使用して、デプロイされた TKG クラスタを更新することはできません。

前提条件

このタスクでは、kubectl edit コマンドを使用する必要があります。このコマンドを実行すると、KUBE_EDITOR または EDITOR 環境変数によって定義されたテキスト エディタで、クラスタ マニフェストが開かれます。ファイルを保存すると、変更が反映されてクラスタが更新されます。kubectl のエディタを構成するには、kubectl のテキスト エディタの構成を参照してください。

手順

1 スーパーバイザー で認証します。

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 ターゲット ワークロード クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 使用可能なストレージ クラスを判別して、使用するクラスを決定するには、次のコマンドを実行します。

```
kubectl describe tanzukubernetescluster CLUSTER-NAME
```

4 次のコマンドを実行して、クラスタのマニフェストを編集します。

v1alpha3 クラスタ：

```
kubectl edit tanzukubernetescluster/CLUSTER-NAME
```

v1beta1 クラスタ：

```
kubectl edit cluster/CLUSTER-NAME
```

5 `storageClass` の値を変更して、マニフェストを編集します。

たとえば、v1alpaha3 クラスタの場合、制御プレーンおよびワーカー ノードに `silver-storage-class` クラスを使用しないようにクラスタのマニフェストを変更します。

```
spec:
  topology:
    controlPlane:
      ...
      storageClass: silver-storage-class
    workers:
      ...
      storageClass: silver-storage-class
```

制御プレーンおよびワーカー ノードに `gold-storage-class` クラスを使用するには、次の手順を実行します。

```
spec:
  topology:
    controlPlane:
      ...
      storageClass: gold-storage-class
    workers:
      ...
      storageClass: gold-storage-class
```

同様に、v1beta1 クラスタをプロビジョニングした場合、クラスタ仕様の `variables.storageclass` の値をストレージ クラスの名前で更新します。

6 マニフェスト ファイルに行った変更内容を保存します。

ファイルを保存すると、kubectl によって変更がクラスタに適用されます。バックグラウンドで、Tanzu Kubernetes Grid は新しいノード仮想マシンをプロビジョニングし、古い仮想マシンをスピンダウンします。

7 kubectl から、マニフェストの編集が正常に記録されたことが報告されているかを確認します。

```
kubectl edit tanzukubernetescluster/tkgs-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkgs-cluster-1 edited
```

注： エラーが表示された場合、またはクラスタ マニフェストが正常に編集されたことが kubectl から報告されない場合は、KUBE_EDITOR 環境変数を使用して、デフォルトのテキスト エディタが適切に構成されていることを確認してください。kubectl のテキスト エディタの構成を参照してください。

8 クラスタが更新されたことを確認します。

v1alpha3 クラスタ：

```
kubectl get tanzukubernetescluster
```

v1beta1 クラスタ：

```
kubectl get cluster
```

仮想マシン クラスの編集による TKG サービス クラスタの更新

クラスタ ノードをホストするために使用されている仮想マシンのクラスを変更することで、TKG サービス クラスタを更新できます。

`kubectl edit` コマンド を使用して `vmClass` の定義を編集することで、TKG サービス クラスタのローリング アップデートを開始できます。変更されたクラスに基づく新しいノードがロールアウトされ、古いノードが停止します。

注： `kubectl apply` コマンドを使用して、デプロイされた TKG クラスタを更新することはできません。

前提条件

このタスクでは、kubectl edit コマンドを使用する必要があります。このコマンドを実行すると、KUBE_EDITOR または EDITOR 環境変数によって定義されたテキスト エディタで、クラスタ マニフェストが開かれます。ファイルを保存すると、変更が反映されてクラスタが更新されます。kubectl のエディタを構成するには、kubectl のテキスト エディタの構成を参照してください。

手順

1 スーパーバイザー で認証します。

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 ターゲット TKG クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context SUPERVISOR-NAMESPACE
```

3 ターゲット TKG クラスタを記述し、仮想マシンのクラスを確認します。

v1alpha3 クラスタ：

```
kubectl describe tanzukubernetescluster CLUSTER-NAME
```

v1beta1 クラスタ：

```
kubectl describe cluster CLUSTER-NAME
```

4 クラスタがプロビジョニングされている vSphere 名前空間 で使用可能な仮想マシン クラスを一覧表示して、記述します。

```
kubectl get virtualmachineclass
```

注： ターゲット仮想マシン クラスは、TKG クラスタがプロビジョニングされている vSphere 名前空間 に関連付けられている必要があります。仮想マシン クラスを vSphere 名前空間 にバインドする方法の詳細については、TKG サービスまたは仮想マシン サービスのドキュメントを参照してください。

5 次のコマンドを実行して、クラスタのマニフェストを編集します。

v1alpha3 クラスタ：

```
kubectl edit tanzukubernetescluster/CLUSTER-NAME
```

v1beta1 クラスタ：

```
kubectl edit cluster/CLUSTER-NAME
```

6 仮想マシン クラス文字列を変更して、マニフェストを編集します。

たとえば、v1alpah3 クラスタを使用している場合、ワーカー ノードに `guaranteed-medium` 仮想マシン クラスを使用しないようにクラスタのマニフェストを変更します。

```
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
```

ワーカーノードに `guaranteed-large` 仮想マシン クラスを使用するには、次の手順を実行します。

```
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
```

同様に、v1beta1 クラスタをプロビジョニングした場合、`variables.vmclass` の値をターゲット仮想マシン クラスに更新します。

7 マニフェスト ファイルに行った変更内容を保存します。

ファイルを保存すると、kubectl によって変更がクラスタに適用されます。バックグラウンドで、TKG コントローラが新しいノード仮想マシンをプロビジョニングし、古い仮想マシンをスピンダウンします。

8 kubectl から、マニフェストの編集が正常に記録されたことが報告されているかを確認します。

```
kubectl edit tanzukubernetescluster/tkgs-cluster-1
tanzukubernetescluster.run.tanzu.vmware.com/tkgs-cluster-1 edited
```

注： エラーが表示された場合、またはクラスタ マニフェストが正常に編集されたことが kubectl から報告されない場合は、KUBE_EDITOR 環境変数を使用して、デフォルトのテキスト エディタが適切に構成されていることを確認してください。kubectl のテキスト エディタの構成を参照してください。

9 クラスタが更新されたことを確認します。

v1alpha3 クラスタ：

```
kubectl get tanzukubernetescluster
```

v1beta1 クラスタ：

```
kubectl get cluster
```

Tanzu CLI を使用した TKG クラスタの更新

TKG クラスタを更新するには、Tanzu CLI を使用して Tanzu Kubernetes リリース バージョンをアップグレードします。

Tanzu CLI を使用して Tanzu Kubernetes リリース バージョンをアップグレードすることで、TKGS クラスタのローリング アップデートを開始できます。

使用方法の詳細については、『Tanzu CLI リファレンス ガイド』を参照してください。

前提条件

TKG サービス クラスタで使用するための Tanzu CLI のインストール。

手順

1 スーパーバイザー で認証します。

2 TKG クラスタを一覧表示します。

```
tanzu cluster list
```

3 TKG クラスタを更新します。

```
tanzu cluster upgrade CLUSTER-NAME --tkr TKR-NAME -n VSPHERE-NAMESPACE
```

ここで、

- `CLUSTER-NAME` は、アップグレードのターゲットとしている TKG クラスタの名前です。
- `TKR-NAME` は、TKR バージョンの文字列です
- `VSPHERE-NAMESPACE` は、TKG クラスタがプロビジョニングされている vSphere 名前空間 の名前です

例：

```
tanzu cluster upgrade tkg-cluster-1 --tkr v1.23.8---vmware.2-tkg.2-zshippable -n tkg2-cluster-ns
```

4 クラスタ アップグレードを確認します。

クラスタがアップグレードされると、次のようなメッセージが表示されます。

```
Cluster 'tkg-cluster-1' successfully upgraded to kubernetes version 'v1.23.8+vmware.2-tkg.2-zshippable'
```

TKG サービス クラスタの自動スケーリング

10

このセクションでは、TKG サービス クラスタの自動スケーリングに関する情報を提供します。

次のトピックを参照してください。

- クラスタの自動スケーリングについて
- kubectl を使用したクラスタ自動スケーラのインストール
- Tanzu CLI を使用したクラスタ自動スケーラのインストール
- Kubectl を使用した自動スケール クラスタのアップグレード
- Tanzu CLI を使用した自動スケール クラスタのアップグレード
- クラスタ自動スケーラのテスト
- クラスタ自動スケーラの削除

クラスタの自動スケーリングについて

クラスタ自動スケーラをデプロイして、ワークロードの要求に基づいて TKG サービス クラスタ内のワーカー ノードの数を自動的に調整できます。

クラスタの自動スケーリングについて

TKG サービス クラスタ自動スケーラは、Kubernetes Cluster Autoscaler の実装です。詳細については、クラスタ自動スケーラのドキュメントを参照してください。

クラスタ自動スケーラは、クラスタ ノードのスケール アウトとスケール インをサポートします。マルチゾーン スーパーバイザー でクラスタを実行している場合、自動スケーラは特定のアベイラビリティ ゾーンに割り当てられたノード プールをスケーリングできます。

クラスタ自動スケーラは、Kubectl または Tanzu CLI を使用してクラスタにインストールする標準パッケージとして提供されます。クラスタ自動スケーラは、サービス アカウントの認証情報を使用して、TKG クラスタ上のデプロイとして実行されます。

自動スケーラ パッケージのマイナー バージョンと TKr のマイナー バージョンの間には、1 対 1 の関係があります。たとえば、TKr 1.27.11 を使用している場合は、自動スケーラの v1.27.2 をインストールする必要があります。バージョンが一致しない場合は、パッケージの調整が失敗します。

クラスタ自動スケーラはワーカー ノードのスケール アウトとスケール インの両方をサポートしますが、一部のタイプのアプリケーションではノードのスケール ダウンが妨げられるため、クラスタ自動スケーラによるノードのスケール ダウンが実行されない場合があります。クラスタ自動スケーラのドキュメントで、「What types of pods can prevent CA from removing a node?」を参照してください。

バージョン要件

クラスタ自動スケーラには、次のバージョン要件があります。

- vSphere の最小バージョンは、vSphere 8 U3
- vSphere 8 の場合、TKr の最小バージョンは TKr 1.27.x
- TKr のマイナー バージョンとクラスタ自動スケーラ パッケージのマイナー バージョンが一致

パッケージ要件

クラスタ自動スケーラは、標準パッケージとして提供されます。パッケージのマイナー バージョンは、使用する TKr のマイナー バージョンと一致する必要があります。たとえば、TKr 1.27.11 を使用している場合は、自動スケーラの v1.27.2 をインストールする必要があります。バージョンが一致しない場合は、パッケージの調整が失敗します。

目的のパッケージは、後続バージョンのリポジトリから見つけることが必要な場合があります。たとえば、自動スケーラの v1.27.2 は、標準パッケージ リポジトリの v2024.4.12 バージョンに含まれています。1.28.x、1.29.x、1.30.x など、それ以降のバージョンの自動スケーラ パッケージは、後続バージョンのリポジトリにあります。すべての標準パッケージ リポジトリを確認するには、次のコマンドを実行します。

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

ワークフロー

クラスタの自動スケーリングを有効にするワークフローの概要は次のとおりです。

1 新しい TKG クラスタを作成するか、既存の TKG クラスタを更新して自動スケーラの注釈を付け、`spec.topology.workers.machinedeployments` のレプリカ フィールドを削除します。

2 作成または更新した TKG クラスタにパッケージ リポジトリをインストールします。

3 作成または更新した TKG クラスタに自動スケーラ パッケージをインストールします。

自動スケーラは、kube-system 名前空間のデプロイとして TKG クラスタにインストールされます。

詳細な手順については、次のトピックを参照してください。

- kubectl を使用したクラスタ自動スケーラのインストール
- Tanzu CLI を使用したクラスタ自動スケーラのインストール

kubectl を使用したクラスタ自動スケーラのインストール

kubectl を使用してクラスタ自動スケーラ パッケージをインストールおよび構成するには、次の手順を参照してください。

要件

次の要件を満たす必要があります。

- vSphere の最小バージョンは、vSphere 8 U3
- vSphere 8 の場合、TKr の最小バージョンは TKr 1.27.x
- TKr のマイナー バージョンとクラスタ自動スケーラ パッケージのマイナー バージョンが一致

注目: 自動スケーラ パッケージのマイナー バージョンと TKr のマイナー バージョンの間には、1 対 1 の関係があります。たとえば、TKr 1.27.11 を使用している場合は、自動スケーラの v1.27.2 をインストールする必要があります。バージョンが一致しない場合は、パッケージの調整が失敗します。

vSphere 名前空間 の構成

前提条件となる次のタスクを実行して、TKG クラスタをプロビジョニングします。

1 環境をインストールするか、vSphere 8 U3 および vSphere 8 用の TKr 1.27.x に更新します。

2 コンテンツ ライブラリを最新の Tanzu Kubernetes リリース で作成またはアップデートします。5 章 TKG サービス クラスタ用 Kubernetes リリースの管理を参照してください。

3 TKG クラスタをホストするための vSphere 名前空間を作成して構成します。6 章 TKG サービス クラスタをホストするための vSphere 名前空間 の構成を参照してください。

4 vSphere 向け Kubernetes CLI Tools をインストールします。

次の例を使用して、コマンド ラインからツールをインストールできます。その他のガイダンスについては、vSphere 向け Kubernetes CLI Tools のインストールを参照してください。

```
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 `kubectl` と `kubectl vsphere` を実行してインストールを確認します。

自動スケーラの注釈を使用した TKG クラスタの作成

次の手順に従って、TKG クラスタを作成します。その他のガイダンスについては、Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。

自動スケーラを使用するには、ここで説明するクラスタ仕様の例に示すように、自動スケーラ ラベルの注釈を使用してクラスタを構成する必要があります。通常のクラスタ プロビジョニングとは異なり、ワーカー ノード レプリカの数はハードコーディングされません。Kubernetes には、自動スケーラの最小サイズと最大サイズの注釈に基づくレプリカのデフォルト ロジックが組み込まれています。これは新しいクラスタであるため、クラスタの作成には最小サイズが使用されます。詳細については、https://cluster-api.sigs.k8s.io/tasks/automated-machine-management/autoscaling を参照してください。

1 kubectl を使用して、スーパーバイザー で認証します。

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-username USERNAME
```

2 クラスタをホストするターゲット vSphere 名前空間にコンテキストを切り替えます。

```
kubectl config use-context tkgs-cluster-namespace
```

3 vSphere 名前空間 で使用可能な仮想マシン クラスを一覧表示します。

使用できるのは、ターゲット vSphere 名前空間にバインドされている仮想マシン クラスのみです。TKG サービス クラスタでの仮想マシン クラスの使用を参照してください。

4 使用可能なパーシステント ボリューム ストレージ クラスを一覧表示します。

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

このコマンドは、vSphere 名前空間に関する詳細情報を返します。これにはストレージ クラスの情報が含まれます。`kubectl describe storageclasses` コマンドを実行すると使用可能なストレージ クラスも返されますが、それには vSphere 管理者の権限が必要です。

5 使用可能な Tanzu Kubernetes リリース を一覧表示します。

```
kubectl get tkr
```

このコマンドは、この vSphere 名前空間で使用可能な TKr とその互換性を返します。5 章 TKG サービス クラスタ用 Kubernetes リリースの管理を参照してください。

6 収集した情報を使用して、必要なクラスタ自動スケーラ構成を含む TKG クラスタ仕様の YAML ファイルを作成します。

- ワーカー ノード プールの `*-min-size` 注釈と `*-max-size` 注釈を使用します。この例では、拡張可能なワーカー ノードの最小数が 3、最大数が 5 です。デフォルトでは、クラスタは 3 つのワーカー ノードで作成されます。
- TKr と自動スケーラ パッケージには、一致するマイナー バージョンを使用します。

- 使用されるクラスタの `metadata.name` と `metadata.namespace` の値は、自動スケーラ パッケージのデフォルト値と一致しています。クラスタ仕様でこれらの値を変更する場合は、`autoscaler-data-values` で変更する必要があります（以下を参照）。

```
#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 クラスタ仕様を適用します。

```
kubectl apply -f cc-autoscaler.yaml
```

8 クラスタの作成を確認します。

```
kubectl get cluster,vm
```

9 クラスタ ノードのバージョンを確認します。

```
kubectl get node
```

TKG クラスタへのパッケージ マネージャのインストール

TKG クラスタがプロビジョニングされたら、クラスタにパッケージ マネージャをインストールしてパッケージ リポジトリを設定します。

1 プロビジョニングした TKG クラスタにログインします。

```
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 Carvel imgpkg ツールをインストールします。

```
wget -O- https://carvel.dev/install.sh > install.sh
sudo bash install.sh
```

3 `imgpkg version` を実行してインストールを確認します。

4 パッケージ リポジトリのバージョンを確認します。

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

5 パッケージ リポジトリをインストールします。リポジトリのバージョンを適宜更新します。

```
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 パッケージ リポジトリを確認します。

```
kubectl get packagerepository -A
NAMESPACE    NAME             AGE     DESCRIPTION
tkg-system   tanzu-standard   2m22s   Reconcile succeeded
```

7 クラスタ自動スケーラ パッケージの有無を確認します。

```
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
...
```

自動スケーラ パッケージのインストール

これで、クラスタ自動スケーラ パッケージをインストールできます。クラスタ自動スケーラは、`kube-system` 名前空間にデプロイとしてインストールされます。

1 `autoscaler.yaml` 構成ファイルを作成します。

- 環境に適した値を使用して仕様の `autoscaler-data-values` セクションを変更することで、自動スケーラをカスタマイズできます。

```
#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 クラスタ自動スケーラ パッケージをインストールします。

```
kubectl apply -f autoscaler.yaml
```

3 自動スケーラ パッケージのインストールを確認します。

```
kubectl get pkgi -A | grep autoscaler
```

予期される結果：

```
tkg-system autoscaler cluster-autoscaler.tanzu.vmware.com 1.27.2+vmware.1-tkg.3 Reconcile
succeeded 3m52s
```

4 自動スケーラのデプロイを確認します。

```
kubectl get pods -n kube-system | grep autoscaler
```

```
cluster-autoscaler-798b65bd9f-bht8n 1/1 Running 0 2m
```

クラスタの自動スケーリングのテスト

クラスタの自動スケーリングをテストするには、アプリケーションをデプロイし、レプリカの数を増やして、負荷を処理するための追加のワーカー ノードがスケール アウトされていることを確認します。

クラスタ自動スケーラのテストを参照してください。

自動スケーリングされたクラスタのアップグレード

自動スケーリングされたクラスタをアップグレードするには、自動スケーラ パッケージを一時停止します。

Kubectl を使用した自動スケール クラスタのアップグレードを参照してください。

Tanzu CLI を使用したクラスタ自動スケーラのインストール

Tanzu CLI を使用してクラスタ自動スケーラ パッケージをインストールおよび構成するには、次の手順を参照してください。

要件

次の要件を満たす必要があります。

- vSphere の最小バージョンは vSphere 8 U3 です（vCenter Server および ESXi ホストを含む）
- vSphere 8 の場合、TKr の最小バージョンは TKr 1.27.x
- TKr のマイナー バージョンとクラスタ自動スケーラ パッケージのマイナー バージョンが一致

注： 自動スケーラ パッケージのマイナー バージョンと TKr のマイナー バージョンの間には、1 対 1 の関係があります。たとえば、TKr 1.27.11 を使用している場合は、自動スケーラの v1.27.2 をインストールする必要があります。バージョンが一致しない場合は、パッケージの調整が失敗します。

vSphere 名前空間 の構成

前提条件となる次のタスクを実行して、TKG クラスタをプロビジョニングします。

1 環境をインストールするか、vSphere 8 U3 および vSphere 8 用の TKr 1.27.x に更新します。

2 コンテンツ ライブラリを最新の Tanzu Kubernetes リリース で作成またはアップデートします。5 章 TKG サービス クラスタ用 Kubernetes リリースの管理を参照してください。

3 TKG クラスタをホストするための vSphere 名前空間を作成して構成します。6 章 TKG サービス クラスタをホストするための vSphere 名前空間 の構成を参照してください。

4 vSphere 向け Kubernetes CLI Tools をインストールします。

次の例を使用して、コマンド ラインからツールをインストールできます。その他のガイダンスについては、vSphere 向け Kubernetes CLI Tools のインストールを参照してください。

```
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 `kubectl` と `kubectl vsphere` を実行してインストールを確認します。

自動スケーラの注釈を使用した TKG クラスタの作成

次の手順に従って、TKG クラスタを作成します。その他のガイダンスについては、Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。

自動スケーラを使用するには、ここで説明するクラスタ仕様の例に示すように、自動スケーラ ラベルの注釈を使用してクラスタを構成する必要があります。通常のクラスタ プロビジョニングとは異なり、ワーカー ノード レプリカの数はハードコーディングされません。Kubernetes には、自動スケーラの最小サイズと最大サイズの注釈に基づくレプリカのデフォルト ロジックが組み込まれています。これは新しいクラスタであるため、クラスタの作成には最小サイズが使用されます。詳細については、https://cluster-api.sigs.k8s.io/tasks/automated-machine-management/autoscaling を参照してください。

1 kubectl を使用して、スーパーバイザー で認証します。

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-username USERNAME
```

2 クラスタをホストするターゲット vSphere 名前空間にコンテキストを切り替えます。

```
kubectl config use-context tkgs-cluster-namespace
```

3 vSphere 名前空間 で使用可能な仮想マシン クラスを一覧表示します。

使用できるのは、ターゲット vSphere 名前空間にバインドされている仮想マシン クラスのみです。TKG サービス クラスタでの仮想マシン クラスの使用を参照してください。

4 使用可能なパーシステント ボリューム ストレージ クラスを一覧表示します。

```
kubectl describe namespace VSPHERE-NAMESPACE-NAME
```

このコマンドは、vSphere 名前空間に関する詳細情報を返します。これにはストレージ クラスの情報が含まれます。`kubectl describe storageclasses` コマンドを実行すると使用可能なストレージ クラスも返されますが、それには vSphere 管理者の権限が必要です。

5 使用可能な Tanzu Kubernetes リリース を一覧表示します。

```
kubectl get tkr
```

このコマンドは、この vSphere 名前空間で使用可能な TKr とその互換性を返します。5 章 TKG サービス クラスタ用 Kubernetes リリースの管理を参照してください。

6 収集した情報を使用して、必要なクラスタ自動スケーラ構成を含む TKG クラスタ仕様の YAML ファイルを作成します。

- ワーカー ノード プールの `*-min-size` 注釈と `*-max-size` 注釈を使用します。この例では、拡張可能なワーカー ノードの最小数が 3、最大数が 5 です。デフォルトでは、クラスタは 3 つのワーカー ノードで作成されます。
- TKr と自動スケーラ パッケージには、一致するマイナー バージョンを使用します。

- 使用されるクラスタの `metadata.name` と `metadata.namespace` の値は、自動スケーラ パッケージのデフォルト値と一致しています。クラスタ仕様でこれらの値を変更する場合は、`autoscaler-data-values` で変更する必要があります（以下を参照）。

```
#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 クラスタ仕様を適用します。

```
kubectl apply -f cc-autoscaler.yaml
```

8 クラスタの作成を確認します。

```
kubectl get cluster,vm
```

9 クラスタ ノードのバージョンを確認します。

```
kubectl get node
```

TKG クラスタでのパッケージ リポジトリの作成

TKG クラスタがプロビジョニングされたら、Tanzu CLI をインストールしてパッケージ リポジトリを設定します。

1 Tanzu CLI をインストールします。

TKG サービス クラスタで使用するための Tanzu CLI のインストールを参照してください。

2 クラスタにログインします。

```
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 パッケージ リポジトリを作成します。

```
#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
```

自動スケーラ パッケージのインストール

クラスタ自動スケーラ パッケージをインストールします。クラスタ自動スケーラは、`kube-system` 名前空間にインストールされます。

1 Tanzu CLI コマンドを使用して、デフォルトの `values.yaml` を生成します。

```
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 パッケージ インストールの `values.yaml` を更新します。

```
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 Tanzu CLI を使用してクラスタ自動スケーラ パッケージをインストールします。

```
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
```

クラスタの自動スケーリングのテスト

クラスタの自動スケーリングをテストするには、アプリケーションをデプロイし、レプリカの数を増やして、追加の負荷を処理するための追加のワーカー ノードがスケール アウトされていることを確認します。

クラスタ自動スケーラのテストを参照してください。

自動スケーリングされたクラスタのアップグレード

自動スケーリングされたクラスタをアップグレードするには、まず自動スケーラ パッケージを一時停止する必要があります。

Tanzu CLI を使用した自動スケール クラスタのアップグレードを参照してください。

Kubectl を使用した自動スケール クラスタのアップグレード

TKG クラスタをアップグレードする前に、自動スケーラを一時停止する必要があります。TKr バージョンのクラスタをアップグレードした後、TKr のマイナー バージョンと一致するように自動スケーラ パッケージのバージョンを更新する必要があります。

要件

このタスクでは、TKG クラスタにクラスタ自動スケーラがインストールされていることを前提としています。kubectl を使用したクラスタ自動スケーラのインストールを参照してください。

クラスタのアップグレード前：自動スケーラの一時停止

自動スケーラがインストールされている TKG クラスタをアップグレードする前に、まず自動スケーラ パッケージを一時停止する必要があります。

1 `autoscaler-data-values.yaml` シークレットで `paused` のブール値を `true` に設定して、クラスタの自動スケーラ パッケージを一時停止します。

```
---
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 `autoscaler-data-values` シークレットに更新を適用します。

```
kubectl apply -f autoscaler-data-values.yaml
```

クラスタのアップグレード

自動スケーラが一時停止したら、クラスタの更新に進みます。

1 TKG クラスタの Kubernetes バージョンをアップグレードします。

TKR バージョンの編集による TKG クラスタの更新を参照してください。

クラスタのアップグレード後：自動スケーラ パッケージのバージョンの更新

クラスタをアップグレードした後、TKr マイナー バージョンと一致するように自動スケーラ パッケージのバージョンを更新し、一時停止を無効にします。

1 対応する自動スケーラ バージョンを選択します。

TKr と自動スケーラ パッケージのマイナー バージョンが一致している必要があります。たとえば、クラスタを TKr v1.28.8 にアップグレードした場合は、自動スケーラ v1.28.x パッケージを使用する必要があります。

2 ターゲット自動スケーラのバージョンを設定し、paused を false にリセットして、自動スケーラのリソースを更新します。

```
#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 自動スケーラ パッケージに更新を適用します。

```
kubectl apply -f autoscaler-package-upgrade.yaml
```

4 自動スケーラ ポッドが kube システムの名前空間内で実行されていることを確認します。

5 クラスタの自動スケーラをテストします。

クラスタ自動スケーラのテスト。

Tanzu CLI を使用した自動スケール クラスタのアップグレード

TKG クラスタをアップグレードする前に、自動スケーラを一時停止する必要があります。TKr バージョンのクラスタをアップグレードした後、TKr のマイナー バージョンと一致するように自動スケーラ パッケージのバージョンを更新する必要があります。

要件

このタスクでは、TKG クラスタにクラスタ自動スケーラがインストールされていることを前提としています。Tanzu CLI を使用したクラスタ自動スケーラのインストールを参照してください。

クラスタのアップグレード前：自動スケーラの一時停止

自動スケーラがインストールされている TKG クラスタをアップグレードする前に、まず自動スケーラ パッケージを一時停止する必要があります。

1 `values.yaml` 構成ファイルで `paused` のブール値を `true` に設定して、クラスタの自動スケーラ パッケージを一時停止します。

```
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 Tanzu CLI を使用してパッケージを更新します。

```
tanzu package installed update cluster-autoscaler-pkgi -n tkg-system --package cluster-
autoscaler.tanzu.vmware.com --values-file values.yaml
```

クラスタのアップグレード

自動スケーラが一時停止したら、クラスタの更新に進みます。

1 TKG クラスタの Kubernetes バージョンをアップグレードします。

TKR バージョンの編集による TKG クラスタの更新を参照してください。

クラスタのアップグレード後：自動スケーラ パッケージのバージョンの更新

クラスタをアップグレードした後、TKr マイナー バージョンと一致するように自動スケーラ パッケージのバージョンを更新し、paused 状態のキーを false にリセットします。

1 対応する自動スケーラ バージョンを選択します。

TKr と自動スケーラ パッケージのマイナー バージョンが一致している必要があります。たとえば、クラスタを TKr v1.28.8 にアップグレードした場合は、自動スケーラ v1.28.0 パッケージを使用する必要があります。

2 Tanzu CLI コマンドを使用して、デフォルトの `values.yaml` を生成します。

```
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 `new-values.yaml` ファイルを新しいパッケージ バージョンに更新し、paused を false にリセットします。

4 Tanzu CLI を使用して、クラスタ自動スケーラのインストールを更新します。

```
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
```

クラスタ自動スケーラのテスト

インストールされているクラスタ自動スケーラをテストするには、次の手順を参照してください。

要件

このタスクでは、TKG クラスタにクラスタ自動スケーラがインストールされていることを前提としています。

- kubectl を使用したクラスタ自動スケーラのインストール
- Tanzu CLI を使用したクラスタ自動スケーラのインストール

クラスタ自動スケーラのテスト

自動スケーラによってワーカー ノードが自動的にスケーリングされるようにするには、アプリケーションをデプロイしてから、デプロイ内のレプリカの数をスケーリングします。ノード リソースが不足すると、自動スケーラはワーカー ノードをスケール アップします。

1 `app.yaml` という名前の次のアプリケーション定義を作成します。

```
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 アプリケーションを作成します。

```
kubectl apply -f app.yaml
```

3 アプリケーションのレプリカをスケール アップして自動スケーラをトリガします。

たとえば、`spec.selector.replicas` の数を 1 からより大きな数に増やして、追加のワーカー ノードが要求されるようにします。

4 アプリケーションを更新します。

```
kubectl apply -f app.yaml
```

5 負荷を処理するために追加のワーカー ノードが作成されることを確認します。

ノード リソースが不足すると、自動スケーラはワーカー ノードの数をスケール アップします。

クラスタ自動スケーラの削除

インストールされているクラスタ自動スケーラを削除するには、次の手順を参照してください。

要件

このタスクでは、TKG クラスタにクラスタ自動スケーラがインストールされていることを前提としています。

- kubectl を使用したクラスタ自動スケーラのインストール
- Tanzu CLI を使用したクラスタ自動スケーラのインストール

Kubectl を使用したクラスタ自動スケーラの削除

1 Kubectl を使用してクラスタ自動スケーラを削除するには、次のコマンドを使用します。

```
kubectl delete -f autoscaler.yaml
```

注： `autoscaler.yaml` という名前は、自動スケーラ パッケージをデプロイするときに使用した名前です。別の名前を使用した場合は、それに合わせてコマンドを修正します。kubectl を使用したクラスタ自動スケーラのインストールを参照してください。

Tanzu CLI を使用したクラスタ自動スケーラの削除

1 Tanzu CLI を使用してクラスタ自動スケーラを削除するには、次のコマンドを使用します。

```
tanzu package installed delete -n tkg-system cluster-autoscaler-pkgi
```

TKG サービス クラスタへの標準パッケージのインストール

11

VMware が提供する一連の標準のオープン ソース アプリケーションは、パッケージとしてバンドルされています。このパッケージを TKG サービス クラスタにインストールして、TKG サービス クラスタの操作を行うことができます。

次のトピックを参照してください。

- TKG サービス クラスタの標準パッケージ
- vSphere 8.x 用の TKr を使用した TKG クラスタへの標準パッケージのインストール
- 標準パッケージ リファレンス
- vSphere 7.x 用の TKr を使用した TKG クラスタへの標準パッケージのインストール

TKG サービス クラスタの標準パッケージ

vSphere IaaS control plane では、TKG サービス クラスタにインストールするための標準パッケージがサポートされます。

TKG サービス クラスタでサポートされているパッケージ

次の表に、vSphere 8.x 用の TKr を使用してプロビジョニングされた TKG サービス クラスタにインストールできる標準パッケージを示します。ターゲット パッケージをインストールする前に、必要な前提条件をすべて満たしておく必要があります。

パッケージ	説明	方法
Cert Manager	証明書の管理	Cert Manager のインストール
Envoy を使用する Contour	Kubernetes Ingress コントローラとリバース プロキシ	Envoy を使用する Contour のインストール
ExternalDNS	Kubernetes サービスの DNS ルックアップ	ExternalDNS のインストール
Fluent Bit	ログ転送	ExternalDNS のインストール
Alertmanager を使用する Prometheus	監視とアラート	Alertmanager を使用する Prometheus のインストール
Grafana	視覚化	Grafana のインストール
Harbor	コンテナ レジストリ	Harbor レジストリのインストール

パッケージ	説明	方法
外部 CSI スナップショット検証 Webhook vSphere PV CSI Webhook	パーシステント ストレージのスナップショット作成	15 章 TKG サービス クラスタでのスナップショットの作成
クラスタ自動スケーラ	クラスタの自動スケーリング	10 章 TKG サービス クラスタの自動スケーリング

vSphere 8.x 用の TKr を使用した TKG クラスタへの標準パッケージのインストール

vSphere 8.x 用の TKr を使用してプロビジョニングされた TKG サービス クラスタに標準パッケージをインストールするには、このセクションを参照してください。

一般的な要件

TKG サービス クラスタに標準パッケージをインストールするには、次の一般的な要件を満たす必要があります。

プラットフォーム要件

これらの指示は、vSphere 8.x の TKr を使用してプロビジョニングされた TKG クラスタへの標準パッケージのインストールに固有のものです。詳細については、TKr リリース ノートを参照してください。

vSphere 7.x の TKr を使用してプロビジョニングされた TKG クラスタに標準パッケージをデプロイする場合は、「vSphere 7.x 用の TKr を使用した TKG クラスタへの標準パッケージのインストール」を参照してください。

リポジトリ要件

vSphere IaaS control plane は、vSphere 8 互換 TKr の TKG クラスタへの標準パッケージのインストールをサポートしています。vSphere 8 TKG サービス クラスタでの Kubernetes リリースの使用 には、Carvel パッケージ システムや Kapp Controller が含まれています。両方のコンポーネントが、TKG ノードのベースになっている TKr イメージの一部として自動的に管理されます。TKr の vSphere との互換性については、TKR リリース ノートを参照してください。

クライアント要件

vSphere 8.x の TKr を使用してプロビジョニングされた TKG クラスタに標準パッケージをインストールするには、Kubectl、kubectl 向けの vSphere プラグイン、Tanzu CLI などの vSphere 向け Kubernetes CLI Tools が必要です。これらのツールをインストールするには、「TKG サービス クラスタ用 CLI ツールのインストール」を参照してください。

ストレージ要件

標準パッケージをデプロイする TKG クラスタは、デフォルトのストレージ クラスを使用してプロビジョニングする必要があります。特に、Prometheus および Grafana パッケージには、デフォルトのストレージ クラスが必要です。デフォルトのストレージ クラスを指定せずに TKG クラスタをプロビジョニングした場合は、既存のストレージ クラスにパッチを適用し、必要な注釈を追加してこのクラスをデフォルトとして指定できます。ストレージ クラスへのパッチ適用を参照してください。

Tanzu パッケージをインストールする TKG クラスタがプロビジョニングされている vSphere 名前空間 のストレージ制限は、パーシステント ボリュームの要求の合計サイズよりも大きくする必要があります。vSphere 名前空間のストレージ割り当ての詳細については、TKG サービス クラスタ向けの vSphere 名前空間 の構成を参照してください。

表 11-1. 標準パッケージのパーシステント ストレージ要件

コンポーネント	TKG 拡張機能	デフォルトのストレージ サイズ
Grafana	Grafana	8 Gi
Prometheus サーバ	Prometheus	8 Gi
Alertmanager	Prometheus	8 Gi
Harbor	Harbor レジストリ	PVC ごとに異なる

TKG クラスタがプロビジョニングされている vSphere 名前空間 のストレージ制限を調整するには、次の手順を実行します。

1 vSphere Client を使用して、vSphere IaaS control plane が有効になっている vCenter Server にログインします。

2 ターゲット Tanzu Kubernetes クラスタがプロビジョニングされている vSphere 名前空間 を選択します。

3 [構成] - [リソースの制限] を選択します。

4 [編集] をクリックします。

5 [ストレージ] 制限を調整して、Prometheus および Grafana 拡張機能に必要なパーシステント ボリューム要求の合計サイズよりも大きくします。

パッケージ リポジトリの作成

次の手順に従って、vSphere 8.x のために TKr を実行している TKG サービス クラスタに標準パッケージ リポジトリを設定します。

要件

パッケージ リポジトリを作成する前に、次の要件を満たします。

- 一般的な要件
- vSphere 向け Kubernetes CLI Tools のインストール
- TKG サービス クラスタで使用するための Tanzu CLI のインストール
- vSphere 8.x 向けの TKr を使用して TKG クラスタをプロビジョニングします。「Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフロー」および TKR リリース ノートを参照してください。

Carvel imgpkg のインストール

Carvel imgpkg (https://carvel.dev/imgpkg/) ツールを使用すると、標準パッケージ リポジトリの使用可能なバージョンを参照できます。リポジトリはパブリックであるため、ログインする必要はありません。

Carvel imgpkg をインストールするには、次の手順を実行します。

1 次のコマンドを使用して imgpkg をインストールします。

```
wget -O- https://carvel.dev/install.sh > install.sh
sudo bash install.sh
```

2 インストールを確認します。

```
imgpkg version
```

結果の例：

```
imgpkg version 0.42.1
```

3 次のコマンドを実行して、リポジトリのバージョンを一覧表示します。

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

このコマンドは、使用可能な標準パッケージ リポジトリのバージョンを返します。

```
Tags

Name
...
v2024.4.12
v2024.4.19
v2024.5.14
v2024.5.16

39 tags

Succeeded
```

パッケージ リポジトリの作成

TKG クラスタにログインし、パッケージ リポジトリを作成します。

1 クラスタにログインします。

```
kubectl vsphere login --server=IP-or-FQDN --vsphere-username USER@vsphere.local --tanzu-
kubernetes-cluster-name CLUSTER --tanzu-kubernetes-cluster-namespace VSPHERE-NS
```

2 パッケージ リポジトリを作成します。

```
tanzu package repository add standard-repo --url projects.registry.vmware.com/tkg/packages/
standard/repo:v2024.5.16 -n tkg-system
```

注： リポジトリのバージョン文字列を、目的のリポジトリのバージョンと一致するように変更します。

3 使用可能なパッケージを一覧表示します。

```
tanzu package available list -n tkg-system
```

注： リポジトリ内のすべてのパッケージが TKG クラスタでサポートされているわけではありません。サポートされているパッケージの公式リスト「TKG サービス クラスタの標準パッケージ」を参照してください。

4 個々のパッケージの使用可能なバージョンを一覧表示します。

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
```

外部 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
```

Cert Manager のインストール

次の手順に従って、vSphere 8.x 用の TKr を実行している TKG サービス クラスタに Cert Manager をインストールします。

Cert Manager について

Cert Manager は、TKG サービス クラスタの証明書管理を提供します。Cert Manager は、ほとんどの標準パッケージ（Contour、ExternalDNS、Prometheus、Harbor など）の前提条件です。

前提条件

次の前提条件を満たす必要があります。

- 一般的な要件.\
- パッケージ リポジトリの作成

Cert Manager のインストール

Cert Manager をインストールするには、次の手順を実行します。

1 使用可能な Cert Manager のバージョンを一覧表示します。

```
tanzu package available get cert-manager.tanzu.vmware.com -n tkg-system
```

注： 要件が異なる場合を除き、通常は最新バージョンを使用する必要があります。

2 Cert Manager 名前空間を作成します。

```
kubectl create ns cert-manager
```

3 Cert Manager をインストールします。

要件を満たすようにターゲット バージョンを調整します。

```
tanzu package install cert-manager -p cert-manager.tanzu.vmware.com -n cert-manager -v
1.12.2+vmware.2-tkg.2
```

4 Cert Manager のインストールを確認します。

```
tanzu package installed list -n cert-manager
```

```
tanzu package installed get -n cert-manager cert-manager
```

5 パッケージのインストールによって作成されたリソース用の Cert Manager 名前空間を確認します。

```
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
```

トラブルシューティング

次のコマンドを使用して、エラー メッセージを確認します。

```
kubectl get pkgi -A
```

```
kubectl describe pkgi -n cert-manager cert-manage
```

Envoy を使用する Contour のインストール

次の手順に従って、vSphere 8.x 用の TKr を実行している TKG サービス クラスタに Envoy を使用する Contour をインストールします。

前提条件

次の前提条件を満たす必要があります。

- 一般的な要件
- Contour パッケージ リファレンス
- パッケージ リポジトリの作成
- Cert Manager のインストール

Contour のデータ値の作成

Contour のインストールを準備するには、データ値ファイルを作成します。

1 使用可能な Contour パッケージのバージョンを一覧表示します。

```
tanzu package available get contour.tanzu.vmware.com -n tkg-system
```

または、kubectl を使用します。

```
kubectl -n tkg-system get packages | grep contour
```

注： 要件が異なる場合を除き、通常は最新バージョンを使用する必要があります。

2 `contour-default-values.yaml` ファイルを生成します。

```
tanzu package available get contour.tanzu.vmware.com/1.28.2+vmware.1-tkg.1 --default-values-file-output contour-data-values.yaml
```

ここで、

- *1.28.2+vmware.1-tkg.1* はターゲット パッケージのバージョンです
- *contour-data-values.yaml* は、生成するデータ値ファイルの名前とパスです

3 `contour-data-values.yaml` ファイルを編集します。

Envoy サービスを `LoadBalancer` に設定して、クラスタ外からのトラフィックが Kubernetes サービスにアクセスできるようにします。手順については、次の例を参照してください。

```
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
```

Contour のインストール

Envoy を使用する Contour Ingress をインストールするには、次の手順を実行します。

1 Contour パッケージの一意の名前空間を作成します。

```
kubectl create ns tanzu-system-ingress
```

2 Contour をインストールします。

要件を満たすようにバージョンを調整します。

```
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 Contour のインストールを確認します。

```
tanzu package installed list -n tanzu-system-ingress
```

4 Contour オブジェクトと Envoy オブジェクトを確認します。

```
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
```

この例では、Envoy サービスの外部 IP アドレスは 10.197.154.69 です。この IP アドレスは、[ワークロード ネットワーク] - [Ingress] に指定された CIDR 範囲から取得されます。この IP アドレス用に新しいロード バランサ インスタンスが作成されます。このロード バランサ用のサーバ プールのメンバーは Envoy ポッドです。Envoy ポッドは、それらが実行されているワーカー ノードの IP アドレスを想定しているため、クラスタ ノードを照会する (`kubectl get nodes -o wide`) ことで、その IP アドレスを確認できます。

トラブルシューティング

必要に応じて、次のトピックを参照してください。

- Contour パッケージ リファレンス。

ExternalDNS のインストール

次の手順に従って、vSphere 8.x 用の TKr を実行している TKG サービス クラスタに ExternalDNS をインストールします。

ExternalDNS について

ExternalDNS を使用すると、Envoy を使用する Contour などの Ingress コンポーネントを使用して、Kubernetes サービス用の DNS レコードを自動的に作成できます。ExternalDNS パッケージは、AWS Route 53、Azure DNS、および RFC2136 準拠の DNS サーバ（BIND など）の各 DNS プロバイダで検証されます。ExternalDNS パッケージのリファレンスも参照してください。

前提条件

次の前提条件を満たす必要があります。

- 一般的な要件。
- パッケージ リポジトリの作成。
- Cert Manager のインストール。
- Envoy を使用する Contour のインストール。

ExternalDNS のデータ値の作成

ExternalDNS のインストールを準備するには、ExternalDNS のデータ値ファイルを作成します。

1 リポジトリで使用可能な ExternalDNS パッケージのバージョンを一覧表示します。

```
tanzu package available get external-dns.tanzu.vmware.com -n tkg-system
```

または、kubectl を使用します。

```
kubectl -n tkg-system get packages | grep external-dns
```

注： 要件が異なる場合を除き、通常は最新バージョンを使用する必要があります。

2 ExternalDNS パッケージのデータ値ファイルを生成します。

```
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
```

ここで、

- *0.13.6+vmware.1-tkg.1* はターゲット パッケージのバージョンです
- *external-dns-data-values.yaml* は、生成するデータ値ファイルの名前とパスです

3 お使いの環境に応じて、データ値をカスタマイズします。

データ値は、ターゲットとしているサポート対象の DNS サーバによって異なります。例については、「ExternalDNS パッケージのリファレンス」を参照してください。

4 必要に応じて、ExternalDNS パッケージが操作する DNS サーバを定義する configmap を作成します。

例については、「ExternalDNS パッケージのリファレンス」を参照してください。

ExternalDNS のインストール

TKG クラスタに ExternalDNS パッケージをインストールするには、次の手順を実行します。

1 ExternalDNS の名前空間を作成します。

```
kubectl create ns tanzu-system-service-discovery
```

2 Tanzu CLI を使用して ExternalDNS パッケージをインストールします。

```
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 Tanzu CLI を使用してパッケージがインストールされていることを確認します。

```
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
```

リファレンス

必要に応じて、次のトピックを参照してください。

- ExternalDNS パッケージのリファレンス。
- サービス検出のための ExternalDNS のインストール

Fluent Bit のインストール

次の手順に従って、vSphere 8.x 用の TKr を実行している TKG サービス クラスタに Fluent Bit をインストールします。

前提条件

次の前提条件を満たす必要があります。

- 一般的な要件
- パッケージ リポジトリの作成
- Cert Manager のインストール
- Fluent Bit ログのサポート対象の転送先。Fluent Bit パッケージのリファレンスを参照してください。

Fluent Bit のデータ値の作成

Fluent Bit のインストールを準備するには、データ値ファイルを作成します。

1 使用可能な Contour パッケージのバージョンを一覧表示します。

```
tanzu package available get fluent-bit.tanzu.vmware.com -n tkg-system
```

または、kubectl を使用します。

```
kubectl -n tkg-system get packages | grep fluent-bit
```

注： 要件が異なる場合を除き、通常は最新バージョンを使用する必要があります。

2 `fluent-bit-data-values.yaml` ファイルを生成します。

```
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 `fluent-bit-data-values.yaml` ファイルを編集して、値を構成します。

例および使用可能なすべてのパラメータのリストについては、「Fluent Bit パッケージのリファレンス」を参照してください。

Fluent Bit のインストール

Fluent Bit パッケージをインストールするには、次の手順を実行します。

1 Fluent Bit の名前空間を作成します。

```
kubectl create ns tanzu-system-logging
```

2 Fluent Bit をインストールします。

```
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 Fluent Bit のインストールを確認します。

```
tanzu package installed list -n tanzu-system-logging
```

```
tanzu package installed get fluent-bit -n tanzu-system-logging
```

4 Fluent Bit オブジェクトを確認します。

```
kubectl -n tanzu-system-logging get all
```

Alertmanager を使用する Prometheus のインストール

次の手順に従って、vSphere 8.x 用の TKr を実行している TKG サービス クラスタに Alertmanager を使用する Prometheus をインストールします。

前提条件

次の前提条件を満たす必要があります。

- 一般的な要件
- パッケージ リポジトリの作成。
- Cert Manager のインストール。

- Envoy を使用する Contour のインストール（Prometheus ダッシュボードへのアクセスに必要）。
- Prometheus パッケージ リファレンス

Prometheus のデータ値の作成

Prometheus のインストールを準備するには、データ値ファイルを作成します。

1 リポジトリの Prometheus パッケージの最新バージョンを取得します。

```
tanzu package available get prometheus.tanzu.vmware.com -n tkg-system
```

または、kubectl を使用します。

```
kubectl -n tkg-system get packages | grep prometheus
```

注： 要件が異なる場合を除き、通常は最新バージョンを使用する必要があります。

2 `prometheus-data-values.yaml` ファイルを生成します。

```
tanzu package available get prometheus.tanzu.vmware.com/2.45.0+vmware.1-tkg.2 --default-values-file-output prometheus-data-values.yaml
```

ここで、

- *2.45.0+vmware.1-tkg.2* はターゲット パッケージのバージョンです
- *prometheus-data-values.yaml* は、生成するデータ値ファイルの名前とパスです

3 `prometheus-data-values.yaml` ファイルを編集し、Prometheus ダッシュボードへのアクセスに必要な次の値を構成します。データ値ファイルの例と構成パラメータの完全なリストについては、「Prometheus パッケージ リファレンス」を参照してください。

パラメータ	説明
`ingress.tlsCertificate.tls.crt`	自己署名 TLS 証明書が Ingress 用に生成されます。オプションで、指定した証明書をオーバーライドで使用することもできます。
`ingress.tlsCertificate.tls.key`	自己署名 TLS プライベート キーが Ingress 用に生成されます。オプションで、指定した証明書をオーバーライドで使用することもできます。
`ingress.enabled`	値を `true` に設定します（デフォルトは false）。
`ingress.virtual_host_fqdn`	値を `prometheus.<your.domain>` に設定します（デフォルトは `prometheus.system.tanzu`）。
`alertmanager.pvc.storageClassName`	vSphere ストレージ ポリシーの名前を入力します。
`prometheus.pvc.storageClassName`	vSphere ストレージ ポリシーの名前を入力します。

Prometheus のインストール

Prometheus パッケージをインストールするには、次の手順を実行します。

1 名前空間を作成します。

```
kubectl create ns tanzu-system-monitoring
```

2 Prometheus をインストールします。

```
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 Prometheus のインストールを確認します。

```
tanzu package installed list -n tanzu-system-monitoring
```

```
tanzu package installed get prometheus -n tanzu-system-monitoring
```

4 Prometheus オブジェクトと Altermanager オブジェクトを確認します。

```
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
```

Prometheus のインストールのトラブルシューティング

`tanzu package install prometheus` 操作で「Failed to get final advertise address: No private IP address found, and explicit IP not provided」というエラーが返された場合は、パッケージ オーバーレイを適用して alertmanager コンポーネントを再構成します。

1 `overlay-alertmanager.yaml` ファイルを作成します。

```
---
#@ 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 Kubectl を使用して、`overlay-alertmanager.yaml` ファイルからシークレットを作成します。

```
kubectl create secret generic alertmanager-overlay -n tkg-system -o yaml --dry-run=client
--from-file=overlay-alertmanager.yaml | kubectl apply -f -
```

3 Kubectl を使用して、Prometheus パッケージにオーバーレイ シークレットで注釈付けします。

```
kubectl annotate PackageInstall prometheus -n tkg-system ext.packaging.carvel.dev/ytt-
paths-from-secret-name.1=alertmanager-overlay
```

4 インストール コマンドをもう一度実行します。

```
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
```

Prometheus ダッシュボードへのアクセス

Prometheus がインストールされたら、次の手順を実行して Prometheus ダッシュボードにアクセスします。

1 `prometheus-data-values.yaml` ファイルの `ingress` セクションのすべての必須フィールドに値が入力されていることを確認します。

```
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 Envoy を使用する Contour ロード バランサのパブリック（外部）IP アドレスを取得します。

Envoy を使用する Contour のインストールを参照してください。

3 使用した Prometheus の FQDN（デフォルトは `prometheus.system.tanzu`）を Envoy ロード バランサの IP アドレスにマッピングする DNS レコードを作成します。

4 ブラウザを使用して Prometheus の FQDN に移動して、Prometheus ダッシュボードにアクセスします。

Grafana のインストール

次の手順に従って、vSphere 8.x 用の TKr を実行している TKG サービス クラスタに Grafana をインストールします。

前提条件

次の前提条件を満たす必要があります。

- 一般的な要件。
- パッケージ リポジトリの作成
- Cert Manager のインストール
- Envoy を使用する Contour のインストール
- Alertmanager を使用する Prometheus のインストール
- Grafana パッケージのリファレンス

Grafana のデータ値の作成

Grafana のインストールを準備するには、データ値ファイルを作成します。

1 リポジトリの Prometheus パッケージの最新バージョンを取得します。

```
tanzu package available get grafana.tanzu.vmware.com -n tkg-system
```

または、kubectl を使用します。

```
kubectl -n tkg-system get packages | grep grafana
```

注： 要件が異なる場合を除き、通常は最新バージョンを使用する必要があります。

2 `prometheus-data-values.yaml` ファイルを生成します。

```
tanzu package available get grafana.tanzu.vmware.com/10.0.1+vmware.1-tkg.2 --default-values-file-output grafana-data-values.yaml
```

ここで、

- *10.0.1+vmware.1-tkg.2* はターゲット パッケージのバージョンです
- *grafana-data-values.yaml* は、生成するデータ値ファイルの名前とパスです

3 `gafana-data-values.yaml` ファイルを編集して、値を更新します。

`ingress.pvc: storageClassName` とその値を追加します。これは、TKG クラスタからアクセス可能な vSphere ストレージ クラスの名前です。

一般的なエラーを回避するには、データ値ファイルからシークレットを削除して、シークレットを手動で作成します。Grafana インストールのトラブルシューティングを参照してください。

ストレージ クラス フィールドが追加され、シークレットが削除された最小限の `gafana-data-values.yaml` を次に示します。その他の例とパラメータの完全なリストについては、「Grafana パッケージのリファレンス」を参照してください。

```
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
```

Grafana のインストール

Grafana パッケージをインストールするには、次の手順を実行します。

1 Grafana の名前空間を作成します。

```
kubectl create ns tanzu-system-dashboards
```

2 Grafana パッケージをインストールします。

```
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 Grafana のインストールを確認します。

```
tanzu package installed list -n tanzu-system-dashboards
```

```
tanzu package installed get grafana -n tanzu-system-dashboards
```

4 Grafana オブジェクトを確認します。

```
kubectl -n tanzu-system-dashboards get all
```

5 Grafana が保持するパーシステント ボリューム要求を確認します。

```
kubectl -n tanzu-system-dashboards get pvc
```

Grafana インストールのトラブルシューティング

「デフォルトの YAML ファイルから Grafana をインストールする場合にシークレットが作成されない」エラーを回避するには、*grafana-data-values.yaml* から `grafana.secret.*` を削除して、次のようにシークレットを手動で作成します。次に、Grafana パッケージを再デプロイします。

```
kubectl create secret generic grafana -n tanzu-system-dashboards  --from-literal=admin=admin
```

Harbor レジストリのインストール

次の手順に従って、vSphere 8.x 用の TKr を実行している TKG サービス クラスタに Harbor コンテナ レジストリをインストールします。

前提条件

次の前提条件を満たす必要があります。

- 一般的な要件
- パッケージ リポジトリの作成
- Cert Manager のインストール
- Envoy を使用する Contour のインストール
- Harbor パッケージ リファレンス

Harbor のデータ値の作成

Harbor のインストールを準備するには、データ値ファイルを作成します。

1 リポジトリの Harbor パッケージの最新バージョンを取得します。

```
tanzu package available get harbor.tanzu.vmware.com -n tkg-system
```

または、kubectl を使用します。

```
kubectl -n tkg-system get packages | grep harbor
```

注： 要件が異なる場合を除き、通常は最新バージョンを使用する必要があります。

2 `harbor-data-values.yaml` ファイルを生成します。

```
tanzu package available get harbor.tanzu.vmware.com/2.9.1+vmware.1-tkg.1 --default-values-file-output harbor-data-values.yaml
```

ここで、

- *2.9.1+vmware.1-tkg.1* はターゲット パッケージのバージョンです
- *harbor-data-values.yaml* は、生成するデータ値ファイルの名前とパスです

3 `harbor-data-values.yaml` ファイルを編集し、次のパラメータの値を更新します。

必要に応じて追加のパラメータを構成します。Harbor パッケージ リファレンスを参照してください。

フィールド	説明
hostname	Harbor 管理コンソールおよびレジストリ サービスにアクセスするための FQDN。「yourdomain.com」を一意のホスト名に置き換えます。
harborAdminPassword	パスワードを強力で一意のものに変更します（インストール後にユーザー インターフェイスで変更することもできます）。
persistence.persistentVolumeClaim.database.storageClass:	vSphere 名前空間の vSphere ストレージ ポリシーの名前を入力します。
persistence.persistentVolumeClaim.jobservice.storageClass:	vSphere 名前空間の vSphere ストレージ ポリシーの名前を入力します。
persistence.persistentVolumeClaim.redis.storageClass:	vSphere 名前空間の vSphere ストレージ ポリシーの名前を入力します。
persistence.persistentVolumeClaim.registry.storageClass:	vSphere 名前空間の vSphere ストレージ ポリシーの名前を入力します。
persistence.persistentVolumeClaim.trivy.storageClass:	vSphere 名前空間の vSphere ストレージ ポリシーの名前を入力します。
tlsCertificate.tlsSecretLabels:	{"managed-by": "vmware-vRegistry"}

Harbor のインストール

次の手順を実行して、Harbor レジストリをインストールします。

1 Harbor の名前空間を作成します。

```
kubectl create ns tanzu-system-registry
```

2 Harbor をインストールします。

```
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 Harbor のインストールを確認します。

```
tanzu package installed get harbor --namespace tanzu-system-registry
```

LoadBalancer タイプの Envoy サービスを使用する Harbor 向けの DNS の構成

前提条件の Envoy を使用する Contour サービスが LoadBalancer を介して公開される場合は、ロード バランサの外部 IP アドレスを取得し、Harbor の FQDN の DNS レコードを作成します。

1 LoadBalancer タイプの Envoy サービスの External-IP アドレスを取得します。

```
kubectl get service envoy -n tanzu-system-ingress
```

返された External-IP アドレスが次の例のように表示されます。

```
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
```

または、次のコマンドを使用して External-IP アドレスを取得することもできます。

```
kubectl get svc envoy -n tanzu-system-ingress -o
jsonpath='{.status.loadBalancer.ingress[0]}'
```

2 Harbor 拡張機能のインストールを確認するには、次の例のように、ロード バランサの External-IP アドレスにマッピングされる Harbor および Notary の FQDN を指定してローカルの /etc/hosts ファイルを更新します。

```
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 Harbor 拡張機能のインストールを確認するために、Harbor にログインします。

4 Envoy サービスのロード バランサの External-IP アドレスを Harbor の FQDN と Notary の FQDN にマッピングする 2 つの CNAME レコードを DNS サーバに作成します。

5 外部 DNS 拡張機能をインストールします。

NodePort タイプの Envoy サービスを使用する Harbor 向けの DNS の構成

前提条件の Envoy を使用する Contour サービスが NodePort を介して公開される場合は、ワーカー ノードの仮想マシンの IP アドレスを取得し、Harbor の FQDN の DNS レコードを作成します。

注： NodePort を使用するには、harbor-data-values.yaml ファイルに正しい port.https 値を指定しておく必要があります。

1 クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

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

2 クラスタ内のノードを一覧表示します。

```
kubectl get virtualmachines
```

3 ワーカー ノードの 1 つを選択し、次のコマンドを使用して記述します。

```
kubectl describe virtualmachines tkg2-cluster-X-workers-9twdr-59bc54dc97-kt4cm
```

4 Vm Ip: 10.115.22.43 などの仮想マシンの IP アドレスを見つけます。

5 Harbor 拡張機能のインストールを確認するには、次の例のように、ワーカー ノードの IP アドレスにマッピングされる Harbor および Notary の FQDN を指定してローカルの `/etc/hosts` ファイルを更新します。

```
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 Harbor 拡張機能のインストールを確認するために、Harbor にログインします。

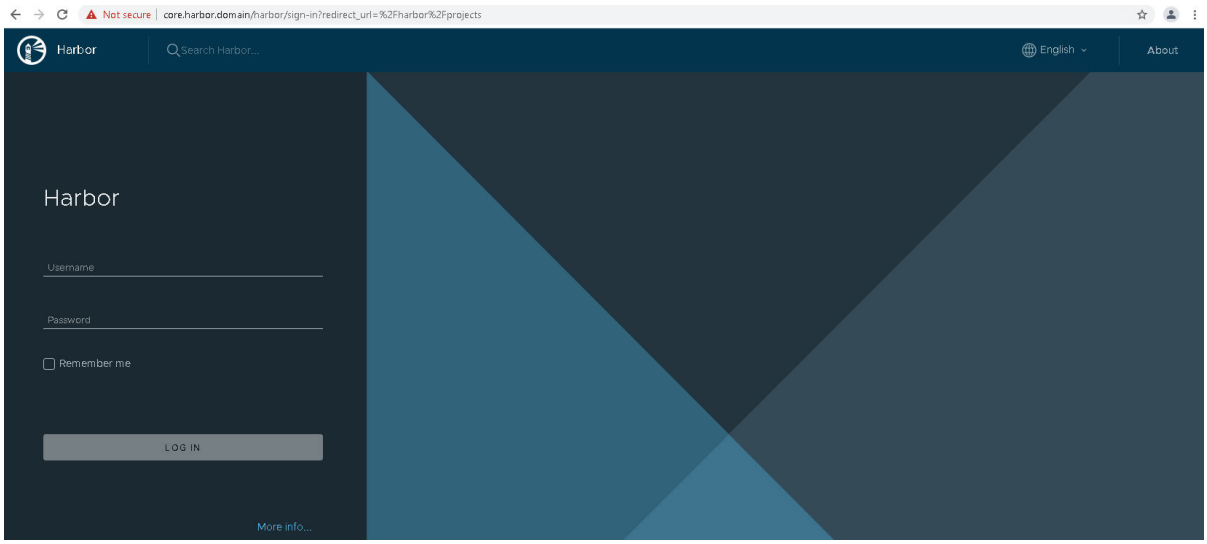
7 ワーカー ノードの IP アドレスを Harbor の FQDN と Notary の FQDN にマッピングする 2 つの CNAME レコードを DNS サーバに作成します。

8 外部 DNS 拡張機能をインストールします。

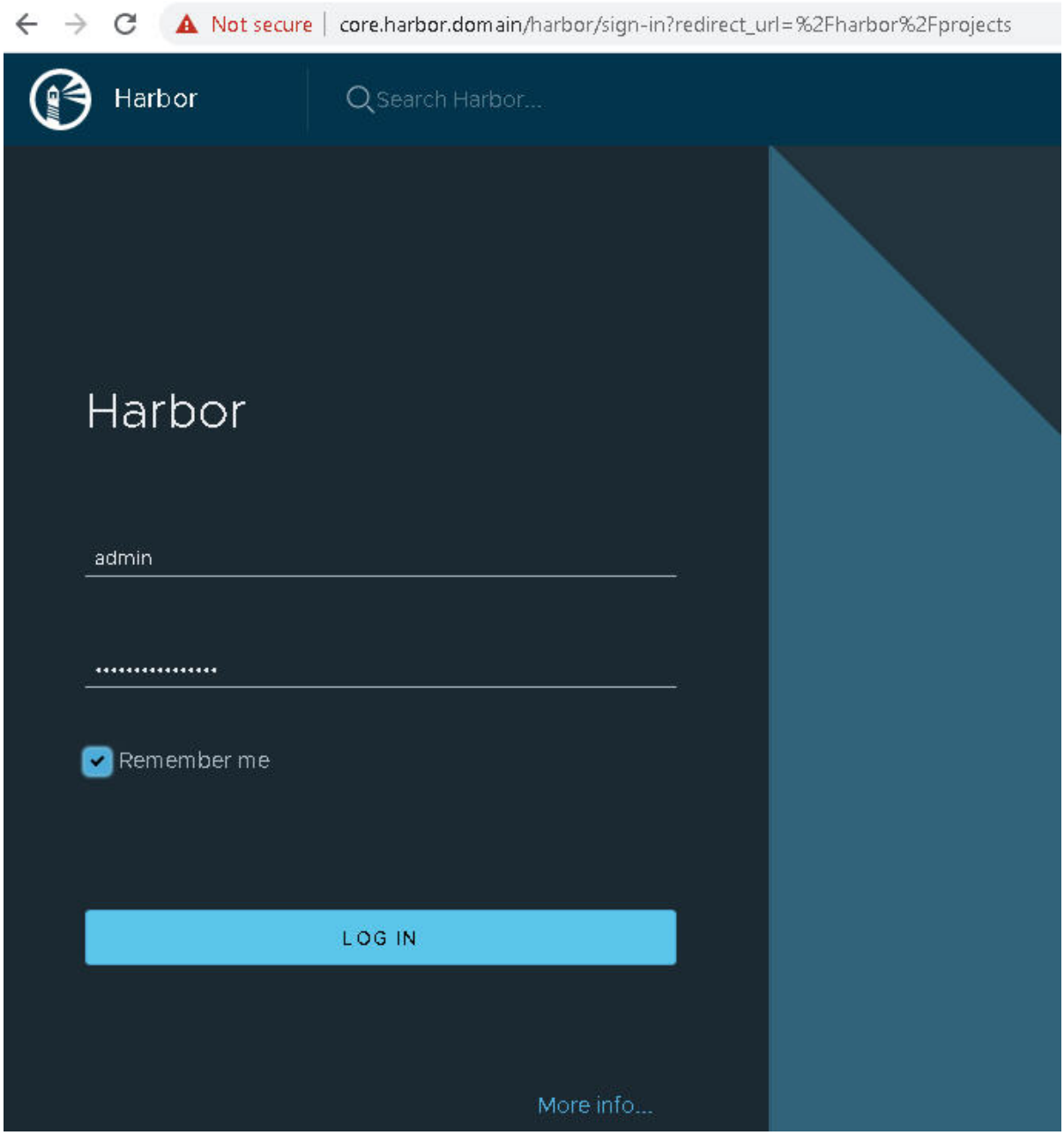
Harbor Web インターフェイスへのログイン

Harbor をインストールして構成したら、ログインして使用を開始します。

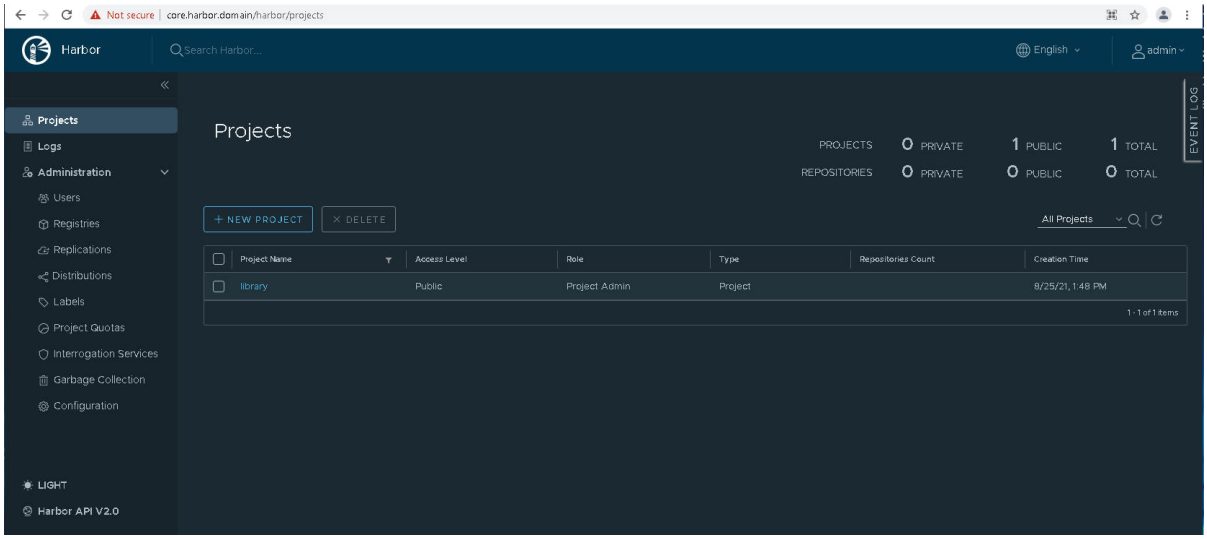
1 Harbor レジストリの Web インターフェイス (`https://core.harbor.domain`)、または使用したホスト名にアクセスします。



2 ユーザー名 **admin** と、生成され、`harbor-data-values.yaml` ファイルに入力したたパスワードを使用して Harbor にログインします。



3 Harbor ユーザー インターフェイスにアクセスできることを確認します。



4 Harbor CA 証明書を取得します。

Harbor インターフェイスで、[プロジェクト] - [ライブラリ] の順に選択するか、[新しいプロジェクト] を作成します。

[レジストリ証明書] をクリックして、Harbor CA 証明書 (`ca.crt`) をダウンロードします。

5 Harbor CA 証明書を Docker クライアントのトラスト ストアに追加して、Harbor レジストリとの間でコンテナ イメージをプッシュおよびプルできるようにします。14 章 TKG サービス クラスタでのプライベート レジストリの使用を参照してください。

6 Harbor の使用方法の詳細については、Harbor のドキュメントを参照してください。

標準パッケージ リファレンス

このセクションでは、TKG サービス クラスタにインストールできる標準パッケージのリファレンス情報を示します。

Contour パッケージ リファレンス

このトピックでは、Envoy を使用する Contour パッケージのリファレンス情報を示します。

Contour と Envory について

Contour (https://projectcontour.io/) は、Envoy リバース HTTP プロキシを含む Kubernetes Ingress コントローラです。Envoy を使用する Contour は、ExternalDNS、Prometheus、Harbor などの他のパッケージで一般的に使用されます。

TKG クラスタに Contour パッケージをインストールするには、次のトピックを参照してください。

- Envoy を使用する Contour のインストール
- #unique_173

Contour のコンポーネント

Contour パッケージには、Contour Ingress コントローラと Envoy リバース HTTP プロキシが含まれています。これらのコンポーネントはコンテナとしてインストールされます。これらのコンテナは、パッケージ リポジトリで指定されているパブリック レジストリからプルされます。

コンテナ	リソースの種類	レプリカ	説明
Envoy	DaemonSet	3	高パフォーマンスのリバース プロキシ
Contour	デプロイ	2	Envoy 用の管理および構成サーバ

Contour のデータ値

`contour-data-values.yaml` のサンプルを以下に示します。

唯一のカスタマイズとして、Envoy サービスのタイプが LoadBalancer（デフォルトは NodePort）になっています。これは、Envoy サービスが Ingress 用クラスタの外部からアクセス可能であることを意味します。

```
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 の構成

Contour のパッケージ構成値は、`contour-data-values.yaml` で設定されます。次の表に、使用可能なパラメータとその説明を示します。

表 11-2. Contour Ingress の構成パラメータ

パラメータ	説明	タイプ	デフォルト
infrastructure_provider	インフラストラクチャ プロバイダ。サポートされている値：vsphere、aws、azure	文字列	必須パラメータ
contour.namespace	contour がデプロイされる名前空間	文字列	tanzu-system-ingress
contour.config.requestTimeout	Envoy に渡されるクライアント要求タイムアウト	time.Duration	0s （詳細は以下のセクションを参照）
contour.config.server.xdsServerType	使用する XDS サーバ タイプ：サポートされている値：contour または envoy	文字列	Null
contour.config.tls.minimumProtocolVersion	Contour がネゴシエートする最小 TLS バージョン	文字列	1.1
contour.config.tls.fallbackCertificate.name	vhost に定義されている SNI と一致しない要求に対するフォールバック証明書が含まれているシークレットの名前	文字列	Null
contour.config.tls.fallbackCertificate.namespace	フォールバック証明書が含まれているシークレットの名前空間	文字列	Null
contour.config.tls.envoyClientCertificate.name	クライアント証明書として使用するシークレットの名前、バックエンド サービスへの TLS 接続用プライベート キー	文字列	Null
contour.config.tls.envoyClientCertificate.namespace	クライアント証明書として使用するシークレットの名前空間、バックエンド サービスへの TLS 接続用プライベート キー	文字列	Null
contour.config.leaderelection.configmapName	contour leaderelection に使用される configmap の名前	文字列	leader-elect
contour.config.leaderelection.configmapNamespace	contour leaderelection configmap の名前空間	文字列	tanzu-system-ingress
contour.config.disablePermitInsecure	ingressroute permitInsecure フィールドを無効にします	boolean	false
contour.config.accesslogFormat	アクセス ログ形式	文字列	envoy
contour.config.jsonFields	ログに記録されるフィールド	文字列の配列	envoy パッケージのドキュメント
contour.config.useProxyProtocol	https://projectcontour.io/guides/proxy-proto/	boolean	false
contour.config.defaultHTTPVersions	Envoy がサービスを提供するためにプログラムする必要がある HTTP バージョン	文字列の配列	"HTTP/1.1 HTTP2"

表 11-2. Contour Ingress の構成パラメータ （続き）

パラメータ	説明	タイプ	デフォルト
contour.config.timeouts.requestTimeout	要求全体のタイムアウト	time.Duration	Null（タイムアウトが無効）
contour.config.timeouts.connectionIdleTimeout	アイドル接続を終了するまでの待機時間	time.Duration	60s
contour.config.timeouts.streamIdleTimeout	アクティビティがない要求またはストリームを終了するまでの待機時間	time.Duration	5m
contour.config.timeouts.maxConnectionDuration	アクティビティに関係なく接続を終了するまでの待機時間	time.Duration	Null（タイムアウトが無効）
contour.config.timeouts.ConnectionShutdownGracePeriod	最初と最後の GOAWAY を送信するまでの待機時間	time.Duration	5s
contour.config.cluster.dnsLookupFamily	HTTPProxy ルートから externalName タイプ サービスへのアップストリーム要求に使用する dns-lookup-family	文字列	Null（サポートされている値：auto、v4、v6）
contour.config.debug	デバッグ機能をオンにします	boolean	false
contour.config.ingressStatusAddress	すべての Ingress リソースのステータスで設定するアドレス	文字列	Null
contour.certificate.duration	contour 証明書の有効期間	time.Duration	8760h
contour.certificate.renewBefore	contour 証明書の更新が必要になるまでの期間	time.Duration	360h
contour.deployment.replicas	contour レプリカの数	整数型	2
contour.image.repository	Contour イメージを含むリポジトリの場所。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg
contour.image.name	contour イメージの名前	文字列	contour
contour.image.tag	Contour イメージ タグ。Contour バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	v1.11.0_vmware.1
contour.image.pullPolicy	Contour イメージ プル ポリシー	文字列	IfNotPresent

表 11-2. Contour Ingress の構成パラメータ （続き）

パラメータ	説明	タイプ	デフォルト
envoy.image.repository	Envoy イメージを含むリポジトリの場所。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg
envoy.image.name	envoy イメージの名前	文字列	envoy
envoy.image.tag	イメージ タグを使用します。Envoy バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	`v1.17.3_vmware.1`
envoy.image.pullPolicy	Envoy イメージ プル ポリシー	文字列	IfNotPresent
envoy.hostPort.enable	ホスト上の envoy ポートを公開するフラグ	boolean	true
envoy.hostPort.http	Envoy HTTP ホスト ポート	整数型	80
envoy.hostPort.https	Envoy HTTPS ホスト ポート	整数型	443
envoy.service.type	envoy を公開するサービスのタイプ。サポートされている値：ClusterIP、NodePort、LoadBalancer	文字列	vSphere の必須パラメータ：NodePort または LoadBalancer、AWS：LoadBalancer、Azure：LoadBalancer
envoy.service.annotations	Envoy サービスの注釈	マップ（キー値）	空のマップ
envoy.service.externalTrafficPolicy	envoy サービスの外部トラフィック ポリシー。サポートされている値：Local、Cluster	文字列	Cluster
envoy.service.nodePort.http	http 要求に使用されるタイプ NodePort のサービスに必要な nodePort	整数型	Null - Kubernetes は動的ノード ポートを割り当てます
envoy.service.nodePort.https	HTTPS 要求に使用されるタイプ NodePort のサービスに必要な nodePort	整数型	Null - Kubernetes は動的ノード ポートを割り当てます
envoy.deployment.hostNetwork	hostNetwork で envoy を実行します	boolean	false
envoy.service.aws.LBType	envoy サービスの公開に使用される AWS LB タイプ。サポートされている値：classic、nlb	文字列	classic
envoy.loglevel	envoy に使用するログ レベル	文字列	info

ファイル ダウンロードのルート タイムアウト

パラメータ `contour.config.requestTimeout` は、Contour ルート タイムアウト期間を定義します。デフォルト値は `0s` です。Contour をファイル転送に使用している場合は、この値の調整が必要になる場合があります。

Contour のドキュメントによると、タイムアウト値が `0s` の場合、Contour は Envoy タイムアウトを使用するように設定されます。Envoy のドキュメントによると、Envoy のデフォルトのタイムアウトは 15 秒です。また、Envoy では、要求から応答までの操作全体がタイムアウト間隔内で完了することが想定されます。

つまり、デフォルトの Contour タイムアウト設定が `0s` の場合、ファイル転送は 15 秒以内に完了する必要があります。これは、サイズが大きいファイルの転送では時間が足りない可能性があります。デフォルトの Envoy のタイムアウトを無効にするには、`contour.config.requestTimeout` の値を `0` に設定します。

ExternalDNS パッケージのリファレンス

このトピックでは、ExternalDNS パッケージのリファレンス情報について説明します。

ExternalDNS について

ExternalDNS により、公開された Kubernetes サービスおよび Ingress が DNS プロバイダに同期されます。

TKG クラスタに ExternalDNS をインストールするには、次のトピックを参照してください。

- vSphere 8x の TKr：ExternalDNS のインストール
- vSphere 7.x の TKr：ExternalDNS のインストール

ExternalDNS のコンポーネント

ExternalDNS パッケージは、表に一覧表示されているコンテナをインストールします。このパッケージは、パッケージ リポジトリで指定されているパブリック レジストリからコンテナをプルします。

コンテナ	リソースの種類	レプリカ	説明
ExternalDNS	DaemonSet	6	DNS ルックアップのための Kubernetes サービスの公開

ExternalDNS データ値

ExternalDNS データ値ファイルは、ExternalDNS コンポーネントとサポートされている DNS プロバイダとのインターフェイスに使用されます。ExternalDNS パッケージは、AWS (Route 53)、Azure DNS、および RFC2136 準拠の DNS サーバ（BIND など）の各 DNS プロバイダで検証されます。

次の例は、RFC2136 準拠の DNS プロバイダ（BIND など）に使用できます。

```
---
# 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
```

次の例は、AWS DNS プロバイダ (Route 53) に使用できます。

```
---
    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
```

次の例は、Azure DNS プロバイダに使用できます。

```
---
    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 の構成

次の表に、ExternalDNS で使用可能な構成パラメータとその説明を示します。その他のガイダンスについては、https://github.com/kubernetes-sigs/external-dns#running-externaldns サイトを参照してください。

表 11-3. 外部 DNS パッケージの構成

パラメータ	説明	タイプ	デフォルト
externalDns.namespace	external-dns がデプロイされる名前空間	文字列	tanzu-system-service-discovery
externalDns.image.repository	external-dns image	文字列	projects.registry.vmware.com/tkg
externalDns.image.name	external-dns の名前	文字列	external-dns
externalDns.image.tag	ExternalDNS のイメージ タグ	文字列	v0.7.4_vmware.1
externalDns.image.pullPolicy	ExternalDNS のイメージ プル ポリシー	文字列	IfNotPresent
externalDns.deployment.annotations	external-dns のデプロイの注釈	map<string,string>	{}
externalDns.deployment.args	コマンドラインを介して external-dns に渡される引数	list<string>	`[]`（必須パラメータ）
externalDns.deployment.env	external-dns に渡す環境変数	list<string>	[]
externalDns.deployment.securityContext	external-dns コンテナのセキュリティ コンテキスト	セキュリティ コンテキスト	{}

表 11-3. 外部 DNS パッケージの構成 （続き）

パラメータ	説明	タイプ	デフォルト
externalDns.deployment.volumeMounts	external-dns コンテナのボリューム マウント	list<VolumeMount>	[]
externalDns.deployment.volumes	external-dns ポッドのボリューム	list<Volume>	[]

configmap の例

次に示す configmap の例では、ExternalDNS が操作できる Kerberos 構成を定義しています。カスタム エントリには、ドメイン/レルム名と kdc/admin_server アドレスが含まれます。

```
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
```

Fluent Bit パッケージのリファレンス

このトピックでは、Fluent Bit パッケージのリファレンス情報について説明します。

Fluent Bit について

Fluent Bit (https://fluentbit.io/) は高速で軽量なログ プロセッサであり、さまざまなソースからアプリケーション データとログを収集し、それらを統一して複数の宛先に送信できます。

Fluent Bit を TKG クラスタからのログのログ フォワーダとして使用できます。ログを保存および分析するためのログ管理サーバをデプロイする必要があります。サポートされるログ サーバは、Syslog、HTTP、Elastic Search、Kafka、Splunk などです。

TKG クラスタに Fluent Bit パッケージをインストールするには、次のトピックを参照してください。

- vSphere 8.x TKr：Fluent Bit のインストール
- vSphere 7.x TKr：#unique_175

Fluent Bit コンポーネント

Fluent Bit パッケージは、表に一覧表示されているコンテナをクラスタにインストールします。このパッケージは、パッケージ リポジトリで指定されているパブリック レジストリからコンテナをプルします。

コンテナ	リソースの種類	レプリカ	説明
Fluent Bit	DaemonSet	6	ログ コレクタ、アグリゲータ、フォワーダ

Fluent Bit のデータ値

次の `fluent-bit-data-values.yaml` の例は、Syslog サーバに使用できます。

```
---
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>"
```

次の `fluent-bit-data-values.yaml` の例は、HTTP エンドポイントに使用できます。

```
---
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"
```

次の `fluent-bit-data-values.yaml` の例は、Elastic Search に使用できます。

```
---
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>"
```

次の `fluent-bit-data-values.yaml` の例は、Kafka に使用できます。

```
---
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>"
```

次の `fluent-bit-data-values.yaml` の例は、Splunk に使用できます。

```
---
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 の構成

構成値は `fluent-bit-data-values.yaml` で設定されています。次の表に、使用可能なパラメータとその説明を示します。

表 11-4. Fluent Bit パッケージの構成

パラメータ	説明	タイプ	デフォルト
logging.namespace	Fluent Bit がデプロイされる名前空間	文字列	tanzu-system-logging
logging.service_account_name	Fluent Bit サービス アカウントの名前	文字列	fluent-bit
logging.cluster_role_name	fluent bit に get、watch、および list の権限を付与するクラスタロールの名前	文字列	fluent-bit-read
logging.image.name	Fluent Bit イメージの名前	文字列	fluent-bit
logging.image.tag	Fluent Bit イメージ タグ。バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	v1.6.9_vmware.1
logging.image.repository	Fluent Bit イメージを含むリポジトリの場所。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg
logging.image.pullPolicy	Fluent bit イメージ プル ポリシー	文字列	IfNotPresent
logging.update_strategy	DaemonSet の更新時に使用される更新方法	文字列	RollingUpdate
tkg.cluster_name	Tanzu Kubernetes クラスタの名前	文字列	Null（必須パラメータ）
tkg.instance_name	スーパーバイザー クラスタ、および 1 つのデプロイ内のすべての Tanzu Kubernetes クラスタで共有される TKG インスタンスのユーザー定義による名前。インストールに関連する任意の名前を使用できます。	文字列	Null（必須パラメータ） **注：** このフィールドは必須ですが、任意に指定できます。ログに表示される名前です。
fluent_bit.log_level	Fluent Bit に使用するログ レベル	文字列	info
fluent_bit.output_plugin	Fluent-Bit が収集する情報をフラッシュするバックエンドを設定します	文字列	Null（必須パラメータ）
fluent_bit.elasticsearch.host	ターゲット Elasticsearch インスタンスの IP アドレスまたはホスト名	文字列	Null（output_plugin が弾性検索の場合の必須パラメータ）
fluent_bit.elasticsearch.port	ターゲット Elasticsearch インスタンスの TCP ポート	整数型	Null（output_plugin が弾性検索の場合の必須パラメータ）

表 11-4. Fluent Bit パッケージの構成 （続き）

パラメータ	説明	タイプ	デフォルト
fluent_bit.elasticsearch.buffer_size	Elasticsearch サービスからの応答の読み取りに使用するバッファ サイズを指定します。False の場合は無制限に設定されます	文字列	False
fluent_bit.elasticsearch.tls	Elasticsearch の TLS のデフォルト設定を指定します	文字列	オフ
fluent_bit.kafka.broker_service_name	Kafka Broker の単一または複数リスト（例：192.168.1.3:9092）	文字列	Null（output_plugin が kafka の場合の必須パラメータ）
fluent_bit.kafka.topic_name	Fluent Bit が Kafka へのメッセージ送信に使用する単一のエントリまたはトピックのカンマ区切りリスト	文字列	Null（output_plugin が kafka の場合の必須パラメータ）
fluent_bit.splunk.host	ターゲット Splunk サーバの IP アドレスまたはホスト名	文字列	Null（output_plugin が splunk の場合の必須パラメータ）
fluent_bit.splunk.port	ターゲット Splunk サーバの TCP ポート	整数型	Null（output_plugin が splunk の場合の必須パラメータ）
fluent_bit.splunk.token	HTTP イベント コレクタ インターフェイスの認証トークンを指定します	文字列	Null（output_plugin が splunk の場合の必須パラメータ）
fluent_bit.http.host	ターゲット HTTP サーバの IP アドレスまたはホスト名	文字列	Null（output_plugin が http の場合の必須パラメータ）
fluent_bit.http.port	ターゲット HTTP サーバの TCP ポート	整数型	Null（output_plugin が http の場合の必須パラメータ）
fluent_bit.http.mode	ターゲット Web サーバの HTTP URI を指定します	文字列	Null（output_plugin が http の場合の必須パラメータ）
fluent_bit.http.header_key_value	HTTP ヘッダー キー/値ペア。複数のヘッダーを設定できます	文字列	Null（output_plugin が http の場合の必須パラメータ）
fluent_bit.http.format	HTTP 要求本文で使用されるデータ形式を指定します	文字列	Null（output_plugin が http の場合の必須パラメータ）
fluent_bit.syslog.host	リモート Syslog サーバのドメインまたは IP アドレス	文字列	Null（output_plugin が syslog の場合の必須パラメータ）
fluent_bit.syslog.port	リモート Syslog サーバの TCP または UDP ポート	整数型	Null（output_plugin が syslog の場合の必須パラメータ）
fluent_bit.syslog.mode	TCP、UDP、TLS からの転送タイプを指定します	文字列	Null（output_plugin が syslog の場合の必須パラメータ）
fluent_bit.syslog.format	HTTP 要求本文で使用されるデータ形式を指定します	文字列	Null（output_plugin が syslog の場合の必須パラメータ）

表 11-4. Fluent Bit パッケージの構成 （続き）

パラメータ	説明	タイプ	デフォルト
host_path.volume_1	ホスト ノードのファイル システムからポッドへのディレクトリ パス（ボリューム 1 の場合）	文字列	/var/log
host_path.volume_2	ホスト ノードのファイル システムからポッドへのディレクトリ パス（ボリューム 2 の場合）	文字列	/var/lib/docker/containers
host_path.volume_3	ホスト ノードのファイル システムからポッドへのディレクトリ パス（ボリューム 3 の場合）	文字列	/run/log
systemd.path	Systemd ジャーナル ディレクトリへのパス	文字列	/var/log/journal

Prometheus パッケージ リファレンス

このトピックでは、Prometheus パッケージのリファレンス情報を示します。

Prometheus および Alertmanager について

Prometheus (https://prometheus.io/) は、システムおよびサービス監視システムです。Prometheus では、構成されたターゲットから指定間隔でメトリックを収集し、ルール式を評価し、結果を表示します。Alertmanager は、何らかの条件が満たされた場合にアラートをトリガするために使用されます。

Prometheus パッケージをインストールするには、次の点に注意してください。

■

■

Prometheus のコンポーネント

Prometheus パッケージは、表に一覧表示されているコンテナを TKG クラスタにインストールします。このパッケージは、パッケージ リポジトリで指定されている VMware パブリック レジストリからコンテナをプルします。

コンテナ	リソースの種類	レプリカ	説明
`prometheus-alertmanager`	デプロイ	1	Prometheus サーバなどのクライアント アプリケーションによって送信されたアラートを処理します。
`prometheus-cadvisor`	DaemonSet	5	実行中のコンテナのリソース使用量とパフォーマンス データを分析して公開します。
`prometheus-kube-state-metrics`	デプロイ	1	ノードのステータスとキャパシティ、レプリカセットのコンプライアンス、ポッド、ジョブ、cron ジョブのステータス、リソース要求、制限を監視します。
`prometheus-node-exporter`	DaemonSet	5	カーネルによって公開されるハードウェアおよび OS メトリックのエクスポータ。

コンテナ	リソースの種類	レプリカ	説明
prometheus-pushgateway	デプロイ	1	スクレイピングできないジョブからメトリックをプッシュできるサービス。
prometheus-server	デプロイ	1	スクレイピング、ルール処理、アラートなどの中核的な機能を提供します。

Prometheus のデータ値

prometheus-data-values.yaml ファイルの例を次に示します。

次の点に注意してください。

- Ingress は有効です (ingress: enabled: true)。
- Ingress は末尾が /alertmanager/ (alertmanagerprefix:) および / (prometheus_prefix:) の URL に対して構成されます。
- Prometheus の FQDN は prometheus.system.tanzu です (virtual_host_fqdn:)。
- Ingress セクションには、独自のカスタム証明書を指定します (tls.crt、tls.key、ca.crt)。
- alertmanager の pvc は 2GiB です。デフォルト ストレージ ポリシーには storageClassName を指定します。
- prometheus の pvc は 20GiB です。vSphere ストレージ ポリシーには storageClassName を指定します。

```
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 の構成

Prometheus の構成は、`prometheus-data-values.yaml` ファイルで設定されます。次の表に、使用可能なパラメータとその説明を示します。

表 11-5. Prometheus 構成パラメータ

パラメータ	説明	タイプ	デフォルト
monitoring.namespace	Prometheus がデプロイされる名前空間	文字列	tanzu-system-monitoring
monitoring.create_namespace	フラグは、monitoring.namespace で指定された名前空間を作成するかどうかを示します	boolean	false
monitoring.prometheus_server.config.prometheus_yaml	Prometheus に渡される Kubernetes クラスタ監視構成の詳細	yaml ファイル	prometheus.yaml
monitoring.prometheus_server.config.alerting_rules_yaml	Prometheus で定義された詳細なアラート ルール	yaml ファイル	alerting_rules.yaml
monitoring.prometheus_server.config.recording_rules_yaml	Prometheus で定義された詳細なレコード ルール	yaml ファイル	recording_rules.yaml
monitoring.prometheus_server.service.type	Prometheus を公開するサービスのタイプ。サポートされている値：ClusterIP	文字列	ClusterIP
monitoring.prometheus_server.enable_alerts.kubernetes_api	Prometheus で Kubernetes API の SLO アラートを有効にする	boolean	true
monitoring.prometheus_server.sc.aws_type	AWS 上の storageclass に定義された AWS タイプ	文字列	gp2
monitoring.prometheus_server.sc.aws_fsType	AWS 上の storageclass に定義された AWS ファイル システム タイプ	文字列	ext4
monitoring.prometheus_server.sc.allowVolumeExpansion	ボリュームの拡張が AWS 上の storageclass に許可されているかどうかを定義します	boolean	true
monitoring.prometheus_server.pvc.annotations	ストレージ クラスの注釈	マップ	{}
monitoring.prometheus_server.pvc.storage_class	パーシステント ボリュームの要求に使用するストレージ クラス。これはデフォルトで null で、デフォルトのプロビジョナが使用されます	文字列	null

表 11-5. Prometheus 構成パラメータ （続き）

パラメータ	説明	タイプ	デフォルト
monitoring.prometheus_server.pvc.accessMode	パーシステント ボリューム要求に対しアクセス モードを定義します。サポートされている値：ReadWriteOnce、ReadOnlyMany、ReadWriteMany	文字列	ReadWriteOnce
monitoring.prometheus_server.pvc.storage	パーシステント ボリューム要求に対しストレージ サイズを定義します	文字列	8Gi
monitoring.prometheus_server.deployment.replicas	prometheus レプリカの数	整数型	1
monitoring.prometheus_server.image.repository	Prometheus イメージを含むリポジトリの場所。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg/prometheus
monitoring.prometheus_server.image.name	Prometheus イメージの名前	文字列	prometheus
monitoring.prometheus_server.image.tag	Prometheus イメージ タグ。バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	v2.17.1_vmware.1
monitoring.prometheus_server.image.pullPolicy	Prometheus イメージ プル ポリシー	文字列	IfNotPresent
monitoring.alertmanager.config.slack_demo	Alertmanager のスラック通知構成	文字列	`slack_demo:   name: slack_demo   slack_configs:   - api_url: https:// hooks.slack.com   channel: '#alertmanager- test'`

表 11-5. Prometheus 構成パラメータ （続き）

パラメータ	説明	タイプ	デフォルト
monitoring.alertmanager.config.email_receiver	Alertmanager の E メール通知構成	文字列	`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	Alertmanager を公開するサービスのタイプ。サポートされている値：ClusterIP	文字列	ClusterIP
monitoring.alertmanager.image.repository	Alertmanager イメージを含むリポジトリの場所。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg/prometheus
monitoring.alertmanager.image.name	Alertmanager イメージの名前	文字列	alertmanager
monitoring.alertmanager.image.tag	Alertmanager イメージ タグ。バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	v0.20.0_vmware.1
monitoring.alertmanager.image.pullPolicy	Alertmanager イメージ プル ポリシー	文字列	IfNotPresent
monitoring.alertmanager.pvc.annotations	ストレージ クラスの注釈	マップ	{}
monitoring.alertmanager.pvc.storage_class	パーシステント ボリュームの要求に使用するストレージ クラス。これはデフォルトで null で、デフォルトのプロビジョナが使用されます。	文字列	null
monitoring.alertmanager.pvc.accessMode	パーシステント ボリューム要求に対しアクセス モードを定義します。サポートされている値：ReadWriteOnce、ReadOnlyMany、ReadWriteMany	文字列	ReadWriteOnce

表 11-5. Prometheus 構成パラメータ （続き）

パラメータ	説明	タイプ	デフォルト
monitoring.alertmanager.pvc.storage	パーシステント ボリューム要求に対しストレージ サイズを定義します	文字列	2Gi
monitoring.alertmanager.deployment.replicas	alertmanager レプリカの数	整数型	1
monitoring.kube_state_metrics.image.repository	kube-state-metircs イメージを含むリポジトリ。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg/prometheus
monitoring.kube_state_metrics.image.name	kube-state-metircs イメージの名前	文字列	kube-state-metrics
monitoring.kube_state_metrics.image.tag	kube-state-metircs イメージ タグ。バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	v1.9.5_vmware.1
monitoring.kube_state_metrics.image.pullPolicy	kube-state-metircs イメージ プル ポリシー	文字列	IfNotPresent
monitoring.kube_state_metrics.deployment.replicas	kube-state-metrics レプリカの数	整数型	1
monitoring.node_exporter.image.repository	node-exporter イメージを含むリポジトリ。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg/prometheus
monitoring.node_exporter.image.name	node-exporter イメージの名前	文字列	node-exporter
monitoring.node_exporter.image.tag	node-exporter イメージ タグ。バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	v0.18.1_vmware.1
monitoring.node_exporter.image.pullPolicy	node-exporter イメージ プル ポリシー	文字列	IfNotPresent
monitoring.node_exporter.hostNetwork	ポッドが `hostNetwork: true` に設定されている場合は、そのポッドで、ネットワークの名前空間とノードのネットワーク リソースを使用できます。	boolean	false
monitoring.node_exporter.deployment.replicas	node-exporter レプリカの数	整数型	1

表 11-5. Prometheus 構成パラメータ （続き）

パラメータ	説明	タイプ	デフォルト
monitoring.pushgateway.image.repository	pushgateway イメージを含むリポジトリ。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg/prometheus
monitoring.pushgateway.image.name	pushgateway イメージの名前	文字列	pushgateway
monitoring.pushgateway.image.tag	pushgateway イメージ タグ。バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	v1.2.0_vmware.1
monitoring.pushgateway.image.pullPolicy	pushgateway イメージ プル ポリシー	文字列	IfNotPresent
monitoring.pushgateway.deployment.replicas	pushgateway レプリカの数	整数型	1
monitoring.cadvisor.image.repository	cadvisor イメージを含むリポジトリ。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg/prometheus
monitoring.cadvisor.image.name	cadvisor イメージの名前	文字列	cadvisor
monitoring.cadvisor.image.tag	cadvisor イメージ タグ。バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	v0.36.0_vmware.1
monitoring.cadvisor.image.pullPolicy	cadvisor イメージ プル ポリシー	文字列	IfNotPresent
monitoring.cadvisor.deployment.replicas	cadvisor レプリカの数	整数型	1
monitoring.ingress.enabled	prometheus および alertmanager に対し ingress を有効または無効にします	boolean	false ingress を使用するには、このフィールドを `true` に設定して、Contour をデプロイします。Prometheus にアクセスするには、`/etc/hosts` をワーカー ノードの IP アドレスにマッピングするエントリを使用して、ローカル `prometheus.system.tanzu` を更新します。

表 11-5. Prometheus 構成パラメータ （続き）

パラメータ	説明	タイプ	デフォルト
monitoring.ingress.virtual_host_fqdn	Prometheus および Alertmanager にアクセスするホスト名	文字列	prometheus.system.tanzu
monitoring.ingress.prometheus_prefix	prometheus のパス プリフィックス	文字列	/
monitoring.ingress.alertmanager_prefix	alertmanager のパス プリフィックス	文字列	/alertmanager/
monitoring.ingress.tlsCertificate.tls.crt	独自の TLS 証明書を使用する場合は、Ingress のオプション証明書。自己署名証明書はデフォルトで生成されます	文字列	Generated cert
monitoring.ingress.tlsCertificate.tls.key	独自の TLS 証明書を使用する場合は、Ingress のオプション証明書プライベート キー。	文字列	Generated cert key

表 11-6. Prometheus_Server Configmap の構成可能フィールド

パラメータ	説明	タイプ	デフォルト
evaluation_interval	ルールを評価する頻度	duration	1m
scrape_interval	ターゲットを取得する頻度	duration	1m
scrape_timeout	取得要求がタイムアウトになるまでの期間	duration	10s
rule_files	ルール ファイルは glob のリストを指定します。ルールとアラートは、一致するすべてのファイルから読み取られます	yaml ファイル	
scrape_configs	取得構成のリスト。	リスト	
job_name	デフォルトで取得されたメトリックに割り当てられたジョブ名	文字列	
kubernetes_sd_configs	Kubernetes サービス検出構成のリスト。	リスト	
relabel_configs	ターゲット再ラベル付け構成のリスト。	リスト	
action	正規表現の一致に基づいて実行するアクション。	文字列	
regex	抽出された値が一致する正規表現。	文字列	
source_labels	ソース ラベルは、既存のラベルから値を選択します。	文字列	
scheme	要求に使用されるプロトコル スキームを構成します。	文字列	

表 11-6. Prometheus_Server Configmap の構成可能フィールド （続き）

パラメータ	説明	タイプ	デフォルト
tls_config	取得要求の TLS 設定を構成します。	文字列	
ca_file	API サーバ証明書を検証する CA 証明書。	ファイル名	
insecure_skip_verify	サーバ証明書の検証を無効にします。	boolean	
bearer_token_file	オプションのベアラー トークン ファイル認証情報。	ファイル名	
replacement	正規表現が一致する場合に正規表現の置き換えが実行される置き換え値。	文字列	
target_label	結果値が置換アクションで書き込まれるラベル。	文字列	

表 11-7. Alertmanager Configmap の構成可能フィールド

パラメータ	説明	タイプ	デフォルト
resolve_timeout	ResolveTimeout は、アラートに EndsAt が含まれていない場合に alertmanager によって使用されるデフォルト値です	duration	5m
smtp_smarthost	E メールが送信される SMTP ホスト。	duration	1m
slack_api_url	スラック Webhook URL。	文字列	global.slack_api_url
pagerduty_url	API 要求の送信先の pagerduty URL。	文字列	global.pagerduty_url
テンプレート	カスタム通知テンプレート定義を読み取るためのファイル	ファイルのパス	
group_by	アラートをラベルでグループ化します	文字列	
group_interval	グループに追加された新しいアラートに関する通知を送信するまでの待機時間を設定	duration	5m
group_wait	アラートのグループに関する通知を最初に送信するまでの待機時間	duration	30s
repeat_interval	通知がアラートに対してすでに正常に送信されている場合に再送信するまでの待機時間	duration	4h
receivers	通知受信者のリスト。	リスト	
severity	インシデントの重要度。	文字列	

表 11-7. Alertmanager Configmap の構成可能フィールド （続き）

パラメータ	説明	タイプ	デフォルト
channel	通知の送信先となるチャネルまたはユーザー。	文字列	
html	E メール通知の HTML 本文。	文字列	
text	E メール通知のテキスト本文。	文字列	
send_resolved	解決済みアラートについて通知するかどうか。	ファイル名	
email_configs	E メール統合の構成	boolean	

ポッドの注釈により、取得プロセスの適切な管理が可能になります。これらの注釈は、ポッド メタデータの一部である必要があります。Services、DaemonSets などのその他のオブジェクトで設定されている場合は無効になります。

表 11-8. Prometheus ポッドの注釈

ポッドの注釈	説明
`prometheus.io/scrape`	デフォルトの構成ではすべてのポッドが取得されます。false に設定した場合、この注釈はポッドを取得プロセスから除外します。
`prometheus.io/path`	メトリック パスが /metrics ではない場合、この注釈を使用して定義します。
`prometheus.io/port`	ポッドをポッドの宣言されたポートではなく、指示されたポートで取得します（宣言がない場合、デフォルトはポートなしターゲットです）。

以下の DaemonSet マニフェストは、ポート 9102 ですべてのポッドを取得するように Prometheus に指示します。

```
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
```

Grafana パッケージのリファレンス

このトピックでは、Grafana パッケージのリファレンス情報を示します。

Grafana について

Grafana (https://grafana.com/) は、オープン ソースの可視化および分析ソフトウェアです。Grafana を使用すると、保存納場所に関係なく、メトリックの照会、可視化、アラート、および確認を行うことができます。Grafana は、アプリケーション データからグラフを作成し、可視化するためのツールを提供します。

TKG クラスタに Grafana パッケージをインストールするには、次のトピックを参照してください。

- vSphere 8.x の TKr：Grafana のインストール
- vSphere 7.x の TKr：#unique_177

Grafana パッケージのコンポーネント

Grafana パッケージは、表に一覧表示されているコンテナをクラスタにインストールします。Grafana パッケージは、パッケージ リポジトリで指定されているパブリック レジストリからコンテナをプルします。

コンテナ	リソースの種類	レプリカ	説明
Grafana	デプロイ	2	データの可視化

Grafana のデータ値

`grafana-data-values.yaml` ファイルの例を以下に示します。このファイルは、次のようにカスタマイズされています。

- Ingress は有効です (ingress: enabled: true)。
- Ingress は末尾が / (prefix:) の URL に対して構成されます。
- Grafana の FQDN は grafana.system.tanzu (virtual_host_fqdn:) です。
- Grafana の PVC は 2 GB で、デフォルトの vSphere storageClass の下に作成されます。
- Grafana ユーザー インターフェイスの（Base64 エンコードの）管理者パスワード (grafana: secret: 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 の構成

Grafana の構成は、`grafana-data-values.yaml` で設定されます。次の表に、使用可能なパラメータとその説明を示します。

表 11-9. Grafana 構成パラメータ

パラメータ	説明	タイプ	デフォルト
monitoring.namespace	Prometheus がデプロイされる名前空間	文字列	tanzu-system-monitoring
monitoring.create_namespace	フラグは、monitoring.namespace で指定された名前空間を作成するかどうかを示します	boolean	false
monitoring.grafana.cluster_role.apiGroups	grafana clusterrole に対して定義された api グループ	リスト	[""]
monitoring.grafana.cluster_role.resources	grafana clusterrole に対して定義されたリソース	リスト	["configmaps", "secrets"]
monitoring.grafana.cluster_role.verbs	clusterrole に対して定義されたアクセス権限	リスト	["get", "watch", "list"]
monitoring.grafana.config.grafana_ini	Grafana 構成ファイルの詳細	構成ファイル	grafana.ini このファイルでは、`grafana_net` URL を使用して Grafana と統合し、たとえばダッシュボードを Grafana.com から直接インポートします。
monitoring.grafana.config.datasource.type	Grafana データソース タイプ	文字列	prometheus
monitoring.grafana.config.datasource.access	アクセス モード。proxy または direct (ユーザー インターフェイスのサーバまたはブラウザ)	文字列	proxy
monitoring.grafana.config.datasource.isDefault	デフォルトの Grafana データソースとしてマーク	boolean	true
monitoring.grafana.config.provider_yaml	grafana ダッシュボード プロバイダを定義する構成ファイル	yaml ファイル	provider.yaml

表 11-9. Grafana 構成パラメータ （続き）

パラメータ	説明	タイプ	デフォルト
monitoring.grafana.service.type	Grafana を公開するサービスのタイプ。サポートされている値：ClusterIP、NodePort、LoadBalancer	文字列	vSphere: NodePort、aws/azure: LoadBalancer
monitoring.grafana.pvc.storage_class	パーシステント ボリューム要求に対しアクセス モードを定義します。サポートされている値：ReadWriteOnce、ReadOnlyMany、ReadWriteMany	文字列	ReadWriteOnce
monitoring.grafana.pvc.storage	パーシステント ボリューム要求に対しストレージ サイズを定義します	文字列	2Gi
monitoring.grafana.deployment.replicas	grafana レプリカの数	整数型	1
monitoring.grafana.image.repository	Grafana イメージを含むリポジトリの場所。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg/grafana
monitoring.grafana.image.name	Grafana イメージの名前	文字列	grafana
monitoring.grafana.image.tag	Grafana イメージ タグ。バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	v7.3.5_vmware.1
monitoring.grafana.image.pullPolicy	Grafana イメージ プル ポリシー	文字列	IfNotPresent
monitoring.grafana.secret.type	Grafana ダッシュボードに定義されたシークレット タイプ	文字列	不透明
monitoring.grafana.secret.admin_user	Grafana ダッシュボードにアクセスするユーザー名	文字列	YWRtaW4= 値は base64 でエンコードされています。デコードには以下を実行します：`echo "xxxxxx" | base64 --decode`
monitoring.grafana.secret.admin_password	Grafana ダッシュボードにアクセスするパスワード	文字列	null
monitoring.grafana.secret.ldap_toml	ldap 認証を使用している場合、ldap 構成ファイルのパス	文字列	""

表 11-9. Grafana 構成パラメータ （続き）

パラメータ	説明	タイプ	デフォルト
monitoring.grafana_init_container.image.repository	grafana init コンテナ イメージを含むリポジトリ。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg/grafana
monitoring.grafana_init_container.image.name	grafana init コンテナ イメージの名前	文字列	k8s-sidecar
monitoring.grafana_init_container.image.tag	grafana init コンテナ イメージ タグ。バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	0.1.99
monitoring.grafana_init_container.image.pullPolicy	grafana init コンテナ イメージ プル ポリシー	文字列	IfNotPresent
monitoring.grafana_sc_dashboard.image.repository	Grafana ダッシュボード イメージを含むリポジトリ。デフォルトは、パブリック VMware レジストリです。プライベート リポジトリを使用している場合は（エアギャップ環境など）、この値を変更します。	文字列	projects.registry.vmware.com/tkg/grafana
monitoring.grafana_sc_dashboard.image.name	grafana ダッシュボード イメージの名前	文字列	k8s-sidecar
monitoring.grafana_sc_dashboard.image.tag	grafana ダッシュボード イメージ タグ。バージョンをアップグレードしている場合は、この値の更新が必要になることがあります。	文字列	0.1.99
monitoring.grafana_sc_dashboard.image.pullPolicy	grafana ダッシュボード イメージ プル ポリシー	文字列	IfNotPresent
monitoring.grafana.ingress.enabled	Grafana に対し Ingress を有効または無効にします	boolean	true
monitoring.grafana.ingress.virtual_host_fqdn	grafana にアクセスするホスト名	文字列	grafana.system.tanzu
monitoring.grafana.ingress.prefix	grafana のパス プリフィックス	文字列	/
monitoring.grafana.ingress.tlsCertificate.tls.crt	独自の TLS 証明書を使用する場合は、Ingress のオプション証明書。自己署名証明書はデフォルトで生成されます	文字列	Generated cert
monitoring.grafana.ingress.tlsCertificate.tls.key	独自の TLS 証明書を使用する場合は、Ingress のオプション証明書プライベート キー。	文字列	Generated cert key

Harbor パッケージ リファレンス

このトピックでは、Harbor レジストリ パッケージのリファレンス情報を示します。

Harbor レジストリについて

Harbor (https://goharbor.io/) は、イメージ リポジトリ、イメージ脆弱性スキャン、およびプロジェクト管理を提供するオープン ソースのコンテナ レジストリ システムです。

スーパーバイザー 上の TKG クラスタに Harbor パッケージをインストールするには、次のトピックを参照してください。

- vSphere 8.x の TKr：Harbor レジストリのインストール
- vSphere 7.x の TKr：#unique_178

Harbor コンポーネント

Harbor パッケージは、表に一覧表示されているコンテナをクラスタにインストールします。このパッケージは、パッケージ リポジトリで指定されているパブリック レジストリからコンテナをプルします。

コンテナ	リソースの種類	レプリカ	説明
harbor-core	デプロイ	1	Envoy 用の管理および構成サーバ
harbor-database	ポッド	1	Postgres データベース
harbor-jobservice	デプロイ	1	Harbor ジョブ サービス
harbor-notary-server	デプロイ	1	Harbor Notary サービス
harbor-notary-signer	デプロイ	1	Harbor Notary
harbor-portal	デプロイ	1	Harbor Web インターフェイス
harbor-redis	ポッド	1	Harbor Redis インスタンス
harbor-registry	デプロイ	2	Harbor コンテナ レジストリ インスタンス
harbor-trivy	ポッド	1	Harbor イメージ脆弱性スキャナ

Harbor のデータ値

Harbor をインストールするための harbor-data-values の例を次に示します。

データ値	説明
hostname: myharbordomain.com	Harbor 管理ユーザー インターフェイスおよびレジストリ サービスにアクセスするための FQDN。
harborAdminPassword: change-it	Harbor 管理者アカウントの初期パスワード。これはインストール時にのみ適用されます。インストール後に Harbor ユーザー インターフェイスまたは API を使用して更新できます。
secretKey: 0123456789ABCDEF	暗号化に使用されるプライベート キー。16 文字の文字列にする必要があります。
database.password: change-it	postgres データベースの初期パスワード。
core.secret: change-it	コア サーバが他のコンポーネントと通信するときに使用されるシークレット。

データ値	説明
`xsrfKey: 0123456789ABCDEF0123456789ABCDEF`	XSRF キー。32 文字の文字列にする必要があります。
`jobservice.secret: change-it`	ジョブ サービスが他のコンポーネントと通信するときに使用されるシークレット。
`registry.secret: change-it`	クライアントおよびレジストリ ストレージ バックエンドからアップロード状態を保護するために使用されるシークレット。
`persistence.persistentVolumeClaim.registry.storageClass: mystorageclass`	ボリュームのプロビジョニングに使用する vSphere ストレージ ポリシーを指定します。
`persistence.persistentVolumeClaim.jobservice.storageClass: mystorageclass`	ボリュームのプロビジョニングに使用する vSphere ストレージ ポリシーを指定します。
`persistence.persistentVolumeClaim.database.storageClass: mystorageclass`	ボリュームのプロビジョニングに使用する vSphere ストレージ ポリシーを指定します。
`persistence.persistentVolumeClaim.redis.storageClass: mystorageclass`	ボリュームのプロビジョニングに使用する vSphere ストレージ ポリシーを指定します。
`persistence.persistentVolumeClaim.trivy.storageClass: mystorageclass`	ボリュームのプロビジョニングに使用する vSphere ストレージ ポリシーを指定します。

Harbor の構成

Harbor の構成は、`harbor-data-values.yaml` ファイルで設定されます。デプロイに最小限必要なフィールドとその説明を次の表に示します。

プロパティ	値	説明
ホスト名	FQDN	Harbor ユーザー インターフェイスにアクセスし、クライアント アプリケーションのレジストリを参照するために指定した FQDN。ドメインは外部 DNS サーバで構成し、Contour によって作成された Envoy サービス IP アドレスに解決されるようにする必要があります。
tlsCertificate.tlsSecretLabels	{"managed-by": "vmware-vRegistry"}	Tanzu Kubernetes Grid が Harbor CA を信頼済みルートとして Tanzu Kubernetes Grid クラスタにインストールする際に使用する証明書。
persistence.persistentVolumeClaim.registry.storageClass	ストレージ ポリシー名。	Harbor レジストリ PVC に使用されるストレージ クラス。
persistence.persistentVolumeClaim.jobservice.storageClass	ストレージ ポリシー名。	Harbor ジョブ サービス PVC に使用されるストレージ クラス。
persistence.persistentVolumeClaim.database.storageClass	ストレージ ポリシー名。	Harbor データベース PVC に使用されるストレージ クラス。
persistence.persistentVolumeClaim.redis.storageClass	ストレージ ポリシー名。	Harbor Redis PVC に使用されるストレージ クラス。
persistence.persistentVolumeClaim.trivy.storageClass	ストレージ ポリシー名。	Harbor Trivy PVC に使用されるストレージ クラス。

vSphere 7.x 用の TKr を使用した TKG クラスタへの標準パッケージのインストール

vSphere 7.x 用のサポート対象の TKr を使用してプロビジョニングされた TKG サービス クラスタにサポート対象の標準パッケージをインストールするには、このセクションを参照してください。

vSphere 7.x の TKr に標準パッケージをインストールするためのワークフロー

このセクションでは、vSphere 7.x の TKr でプロビジョニングされている TKG クラスタに標準パッケージをインストールする手順について説明します。

要件

これらの手順は、vSphere 7.0.3.6 の TKr v1.27.10 および vSphere 8.0.1.1 の TKr v1.27.10 で検証されています。公開時点では、このリリースが vSphere 7.x で使用可能な最新の TKr でした。vSphere 7.x の TKr は vSphere 8.x で実行できますが、vSphere 7.x から vSphere 8.x にアップグレードする目的に限られます。

次の前提条件を満たす必要があります。

- ワークロード管理が有効
- スーパーバイザー がデプロイ済み
- vSphere 名前空間 が作成済み

  TKG サービス クラスタをホストするための vSphere 名前空間 の作成を参照してください。
- vSphere 向け Kubernetes CLI Tools がインストールされている Linux クライアント

  vSphere 向け Kubernetes CLI Tools のインストールを参照してください。

注： vSphere 8.x の TKr でプロビジョニングされた TKG クラスタを使用している場合、標準パッケージのインストール手順については、次のドキュメントを参照してください。vSphere 8.x 用の TKr を使用した TKG クラスタへの標準パッケージのインストール Tkr バージョンの詳細については、リリース ノートを参照してください。

TKG クラスタの作成

標準パッケージをホストするための TKG クラスタを作成します。

1 TKG クラスタを作成します。

  Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。

  TKr v1.27.10 の Photon エディションのクラスタ仕様の例。

```
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
```

TKr v1.27.10 の Ubuntu エディションのクラスタ仕様の例。

```
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
```

Carvel パッケージ マネージャのインストール

Carvel パッケージ マネージャをインストールします。

1 TKG クラスタにログインします。

```
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 Carvel パッケージ マネージャをインストールします。

```
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 インストールを確認します。

```
imgpkg version
```

Kapp Controller のインストール

vSphere 7.x 用の TKr への Kapp Controller のインストールを参照してください。

パッケージ リポジトリの追加

目的のパッケージ リポジトリ バージョンを追加します。

1 最新のリポジトリ タグを一覧表示します。

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/repo
```

2 `packagerepo.yaml` を作成します。

ターゲット バージョンと一致するようにリポジトリのバージョンを更新します。

```
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 パッケージ リポジトリをインストールします。

```
kubectl apply -f packagerepo.yaml
```

予期される結果：

```
packagerepository.packaging.carvel.dev/tanzu-standard created
```

4 パッケージ リポジトリを確認します。

```
kubectl get packagerepositories -A
```

予期される結果：

```
NAMESPACE    NAME              AGE    DESCRIPTION
tkg-system   tanzu-standard    3m9s   Reconcile succeeded
```

Cert Manager のインストール

vSphere 7.x 用の TKr への Cert Manager のインストールを参照してください。

Envoy を使用する Contour のインストール

vSphere 7.x 用の TKr への Contour のインストールを参照してください。

ExternalDNS のインストール

vSphere 7.x 用の TKr への ExternalDNS のインストールを参照してください。

ログ転送のための Fluent Bit のインストール

vSphere 7.x 用の TKr への Fluent Bit のインストールを参照してください。

Prometheus のインストール

vSphere 7.x 用の TKr への Prometheus のインストールを参照してください。

Grafana のインストール

vSphere 7.x 用の TKr への Grafana のインストールを参照してください。

Harbor のインストール

vSphere 7.x 用の TKr への Harbor のインストールを参照してください。

vSphere 7.x 用の TKr への Kapp Controller のインストール

vSphere 7.x 用の TKr を使用してプロビジョニングされた TKG クラスタに Kapp Controller をインストールするには、次の手順を参照してください。

前提条件

vSphere 7.x の TKr に標準パッケージをインストールするためのワークフローを参照してください。

Kapp Controller のインストール

重要： これらの手順は、vSphere 7.x 用の TKr に固有のものです。vSphere 8.x 用の TKr には、Kapp Controller パッケージがすでに含まれています。vSphere 8.x 用の TKr には Kapp Controller を手動でインストールしないでください。

Kapp Controller をインストールします。

1 Kapp Controller ポッドを実行するためのバインドを作成します。

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding --
clusterrole=cluster-admin --group=system:authenticated
```

予期される結果：

```
clusterrolebinding.rbac.authorization.k8s.io/default-tkg-admin-privileged-binding created
```

2 `kapp-controller.yaml` を準備します。

詳細については、ドキュメントを参照してください。

3 Kapp Controller をインストールします。

```
kubectl apply -f kapp-controller.yaml
```

4 Kapp Controller のインストールを確認します。

```
kubectl get all -n tkg-system
```

結果の例：

```
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 Carvel カスタム リソースを確認します。

```
kubectl get crd | grep carvel
```

結果の例：

```
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

次の `kapp-controller.yaml` には、必要な `securityContext` の設定が含まれています。

```
---
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
```

vSphere 7.x 用の TKr への Cert Manager のインストール

vSphere 7.x 用の TKr に Cert Manager をインストールするには、次の手順を参照してください。

前提条件

vSphere 7.x の TKr に標準パッケージをインストールするためのワークフローを参照してください。

Cert Manager のインストール

Cert Manager をインストールします。

1 使用可能な Cert Manager パッケージのバージョンを一覧表示します。

```
kubectl -n tkg-system get packages | grep cert-manager
```

2 ターゲット バージョンを使用して `cert-manager.yaml` を作成します。

cert-manager.yaml を参照してください。

3 Cert Manager をインストールします。

```
kubectl apply -f cert-manager.yaml
```

予期される結果：

```
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 Cert Manager のインストールを確認します。

```
kubectl get pkgi -A
```

予期される結果：

```
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 Cert Manager ポッドを確認します。

```
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

Cert Manager をインストールするには、次の `cert-manager.yaml` の例を参照してください。ターゲット パッケージのバージョンと一致するようにバージョン変数を更新します。

```
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
```

vSphere 7.x 用の TKr への Contour のインストール

vSphere 7.x 用の TKr を使用してプロビジョニングされた TKG クラスタに標準パッケージをインストールするには、次の手順を参照してください。

前提条件

vSphere 7.x の TKr に標準パッケージをインストールするためのワークフローを参照してください。

Envoy を使用する Contour のインストール

Envoy サービスを使用する Contour Ingress をインストールします。

1 リポジトリで使用可能な Contour のバージョンを一覧表示します。

```
kubectl get packages -n tkg-system | grep contour
```

2 `contour.yaml` 仕様を作成します。

#unique_187/unique_187_Connect_42_GUID-CC995CF8-0F4B-4D92-A782-A3832C0EA5AE を参照してください。

3 必要に応じて、環境用に `contour-data-values` をカスタマイズします。

Contour パッケージ リファレンスを参照してください。

4 Contour をインストールします。

```
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 Contour パッケージのインストールを確認します。

```
kubectl get pkgi -A
```

6 Contour オブジェクトを確認します。

```
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
```

Contour パッケージでは、2 つの Contour ポッドと 3 つの Envoy ポッドがインストールされます。Contour と Envoy の両方がサービスとして公開されます。この例では、Envoy サービスの外部 IP アドレスは 10.197.154.69 です。この IP アドレスは、[ワークロード ネットワーク] - [Ingress] に指定された CIDR から取得されます。この IP アドレス用にロード バランサ インスタンスが作成されます。このロード バランサ用のサーバ プールのメンバーは Envoy ポッドです。Envoy ポッドは、それらが実行されているワーカー ノードの IP アドレスを想定しています。クラスタ ノードを照会する (`kubectl get nodes -o wide`) ことで、その IP アドレスを確認できます。

contour.yaml

次の `contour.yaml` を使用して、Envoy を使用する Contour をインストールします。ターゲット パッケージのバージョンと一致するようにバージョン変数を更新します。

```
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
```

vSphere 7.x 用の TKr への ExternalDNS のインストール

vSphere 7.x 用の TKr を使用してプロビジョニングされた TKG クラスタに ExternalDNS をインストールするには、次の手順を参照してください。

前提条件

vSphere 7.x の TKr に標準パッケージをインストールするためのワークフローを参照してください。

ExternalDNS のインストール

vSphere 7.x 用の TKr を使用してプロビジョニングされた TKG クラスタに ExternalDNS をインストールします。

1 リポジトリで使用可能な ExternalDNS バージョンを一覧表示します。

```
kubectl get packages -n tkg-system | grep external-dns
```

2 ExternalDNS 名前空間を作成します。

```
kubectl create namespace tanzu-system-service-discovery --dry-run=client -o yaml | kubectl apply -f -
```

3 名前空間にセキュリティ状態を設定します。

```
kubectl label namespace tanzu-system-service-discovery pod-security.kubernetes.io/enforce=privileged
```

4 バインド デプロイの YAML を準備します。

bind-deployment.yaml を参照してください。

5 BIND DNS サーバをデプロイします。

```
kubectl apply -n tanzu-system-service-discovery -f bind-deployment.yaml
```

6 ExternalDNS デプロイの YAML を準備します。

external-dns-deploy.yaml を参照してください。

7 `external-dns-default-values.yaml` ファイルを使用してシークレットを作成します。

```
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 シークレットを確認します。

```
kubectl get secret external-dns-default-values -n tkg-system
```

```
kubectl get secret external-dns-default-values -n tkg-system -oyaml
```

9 ExternalDNS パッケージ インストールの YAML を準備します。

external-dns-packageinstall.yaml を参照してください。

10 バインドを構成します。

```
sed -i "s/--rfc2136-host=[0-9.]\+/--rfc2136-host=$svcip/g" external-dns-packageinstall.yaml
```

11 外部 DNS パッケージを作成します。

```
kubectl apply -f external-dns-packageinstall.yaml
```

12 ExternalDNS のインストールを確認します。

```
kubectl get all -n tanzu-system-service-discovery
```

bind-deployment.yaml

`bind-deployment.yaml` の例。

```
---
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

`external-dns-deploy.yaml` の例。

```
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

次の例は、BIND に使用できます。必要に応じて、パッケージのバージョンを更新します。

```
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
```

次の例は、AWS DNS プロバイダ (Route 53) に使用できます。必要に応じて、パッケージのバージョンを更新します。

```
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
```

次の例は、Azure DNS プロバイダに使用できます。必要に応じて、パッケージのバージョンを更新します。

```
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
```

vSphere 7.x 用の TKr への Fluent Bit のインストール

vSphere 7.x 用の TKr を使用してプロビジョニングされた TKG クラスタに Fluent Bit をインストールするには、次の手順を参照してください。

前提条件

vSphere 7.x の TKr に標準パッケージをインストールするためのワークフローを参照してください。

Fluent Bit のインストール

ログ転送のための Fluent Bit をインストールします。

1 リポジトリで使用可能な Fluent Bit バージョンを一覧表示します。

```
kubectl -n tkg-system get packages | grep fluent-bit
```

2 名前空間を作成します。

```
kubectl create ns tanzu-system-logging
```

3 PSA の名前空間にラベルを付けます。

```
kubectl label ns fluentbit-logging pod-security.kubernetes.io/enforce=privileged
```

または、次の手順を実行します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: fluentbit-logging
---
apiVersion: v1
kind: Namespace
metadata:
  name: fluentbit-logging
  labels: pod-security.kubernetes.io/enforce: privileged
```

4 `fluentbit.yaml` を準備します。

詳細については、ドキュメントを参照してください。

5 お使いの環境に応じて、`fluentbit-data-values` をカスタマイズします。

構成パラメータについては、「Fluent Bit パッケージのリファレンス」を参照してください。

6 Fluent Bit をインストールします。

```
kubectl apply -f fluentbit.yaml
```

7 Fluent Bit のインストールを確認します。

```
kubectl get all -n fluentbit-logging
```

fluentbit.yaml

次の例は、Syslog エンドポイントに使用できます。必要に応じて、パッケージのバージョンを更新します。

```
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
```

vSphere 7.x 用の TKr への Prometheus のインストール

vSphere 7.x 用の TKr を使用してプロビジョニングされた TKG クラスタに Prometheus をインストールするには、次の手順を参照してください。

前提条件

vSphere 7.x の TKr に標準パッケージをインストールするためのワークフローを参照してください。

Prometheus のインストール

Alertmanager を使用する Prometheus をインストールします。

1 リポジトリで使用可能な Prometheus パッケージのバージョンを一覧表示します。

```
kubectl get packages -n tkg-system | grep prometheus
```

2 Prometheus 名前空間を作成します。

```
kubectl create ns tanzu-system-monitoring
```

3 Prometheus 名前空間で PSA を構成します。

```
kubectl label ns prometheus-monitoring pod-security.kubernetes.io/enforce=privileged
```

```
kubectl get ns prometheus-monitoring -oyaml|grep privileged
```

4 `prometheus-data-values.yaml` ファイルを作成します。

を参照してください。

5 `prometheus-data-values.yaml` を入力として使用してシークレットを作成します。

注： `prometheus-data-values` はサイズが大きいため、Prometheus YAML 仕様にシークレットを含めるのではなく、シークレットを個別に作成する方がエラーが発生する確率が低くなります。

```
kubectl create secret generic prometheus-data-values --from-file=values.yaml=prometheus-data-values.yaml -n tkg-system
```

```
secret/prometheus-data-values created
```

6 シークレットを確認します。

```
kubectl get secrets -A
```

```
kubectl describe secret prometheus-data-values -n tkg-system
```

7 必要に応じて、環境用に `prometheus-data-values` をカスタマイズします。

Prometheus の構成を参照してください。

`prometheus-data-values.yaml` を更新する場合は、次のコマンドを使用してシークレットを置き換えます。

```
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 `prometheus.yaml` 仕様を作成します。

vSphere 7.x 用の TKr への Prometheus のインストールを参照してください。

9 Prometheus をインストールします。

```
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 Prometheus パッケージのインストールを確認します。

```
kubectl get pkgi -A
```

11 Prometheus オブジェクトを確認します。

```
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 Prometheus PVC を確認します。

```
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
```

Prometheus ダッシュボードへのアクセス

Prometheus がインストールされたら、次の手順を実行して Prometheus ダッシュボードにアクセスします。

1 `prometheus-data-values.yaml` ファイルの `ingress` セクションのすべての必須フィールドに値が入力されていることを確認します。

```
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 Envoy を使用する Contour ロード バランサのパブリック（外部）IP アドレスを取得します。

```
kubectl -n tanzu-system-ingress get all
```

3 Prometheus Web インターフェイスを起動します。

```
kubectl get httpproxy -n tanzu-system-monitoring
```

FQDN は、Envoy サービスのパブリック IP アドレスで使用できる必要があります。

```
NAME                    FQDN                        TLS SECRET       STATUS   STATUS
DESCRIPTION
prometheus-httpproxy    prometheus.system.tanzu     prometheus-tls   valid    Valid  HTTPProxy
```

4 Prometheus の FQDN を Envoy ロード バランサの外部 IP アドレスにマッピングする DNS レコードを作成します。

5 ブラウザを使用して Prometheus の FQDN に移動して、Prometheus ダッシュボードにアクセスします。

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\
      - 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    \
      \  \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  \
      \  \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             \
      \   vector(0)\n              )\n              +\n
sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"\
      LIST|GET\",scope=\"namespace\",le=\"0.5\"}[30d]))\n              +\n       \
      \       sum(increase(apiserver_request_duration_seconds_bucket{verb=~\"LIST|GET\"\
      ,scope=\"cluster\",le=\"5\"}[30d]))\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          # 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

`prometheus.yaml` 仕様は `prometheus-data-values` シークレットを参照します。

```
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
```

vSphere 7.x 用の TKr への Grafana のインストール

vSphere 7.x 用の TKr を使用してプロビジョニングされた TKG クラスタに Grafana をインストールするには、次の手順を参照してください。

前提条件

vSphere 7.x の TKr に標準パッケージをインストールするためのワークフローを参照してください。

Grafana のインストール

Grafana をインストールします。

1 リポジトリで使用可能な Grafana バージョンを一覧表示します。

```
kubectl get packages -n tkg-system | grep grafana
```

2 Grafana 名前空間を作成します。

```
kubectl create ns tanzu-system-dashboards
```

3 名前空間の PSA ラベルを作成します。

```
kubectl label namespace tanzu-system-dashboards pod-security.kubernetes.io/
enforce=privileged
```

4 または、`ns-grafana-dashboard.yaml` ファイルを使用して Grafana 名前空間とラベルを宣言によって作成します。

```
apiVersion:
v1kind: Namespace
metadata:
  name: grafana-dashboard
---
apiVersion: v1
kind: Namespace
```

```
metadata:
  name: tanzu-system-dashboards
  labels:
    pod-security.kubernetes.io/enforce: privileged
```

5 `grafana-data-values.yaml` を作成します。

Grafana パッケージのリファレンスを参照してください。

6 `grafana-data-values.yaml` ファイルを入力として使用してシークレットを作成します。

```
kubectl create secret generic grafana-data-values --from-file=values.yaml=grafana-data-values.yaml -n tkg-system
```

```
secret/grafana-data-values created
```

7 シークレットを確認します。

```
kubectl get secrets -A
```

```
kubectl describe secret grafana-data-values -n tkg-system
```

8 必要に応じて、環境用に `grafana-data-values` をカスタマイズします。

Grafana パッケージのリファレンスを参照してください。

データ値を更新する場合は、次のコマンドを使用してシークレットを更新します。

```
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 `grafana.yaml` 仕様を作成します。

vSphere 7.x 用の TKr への Grafana のインストールを参照してください。

10 Grafana をインストールします。

```
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 Grafana パッケージのインストールを確認します。

```
kubectl get pkgi -A | grep grafana
```

12 Grafana オブジェクトを確認します。

```
kubectl get all -n tanzu-system-dashboards
```

Envoy LoadBalancer を使用した Grafana ダッシュボードへのアクセス

LoadBalancer タイプの前提条件である Contour Envoy サービスがデプロイされていて、Grafana 構成ファイル内で指定されている場合は、ロード バランサの外部 IP アドレスを取得し、Grafana FQDN の DNS レコードを作成します。

1 LoadBalancer タイプの Envoy サービスの `External-IP` アドレスを取得します。

```
kubectl get service envoy -n tanzu-system-ingress
```

返された `External-IP` アドレスが次の例のように表示されます。

```
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
```

または、次のコマンドを使用して `External-IP` アドレスを取得することもできます。

```
kubectl get svc envoy -n tanzu-system-ingress -o
jsonpath='{.status.loadBalancer.ingress[0]}'
```

2 Grafana 拡張機能のインストールを確認するには、次の例のように、ロード バランサの `External-IP` アドレスにマッピングされる Grafana の FQDN を指定してローカルの `/etc/hosts` ファイルを更新します。

```
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 `https://grafana.system.tanzu` に移動して、Grafana ダッシュボードにアクセスします。

サイトでは自己署名証明書が使用されるため、ダッシュボードにアクセスする前にブラウザ固有のセキュリティ警告を確認しなければならない場合があります。

4 本番環境にアクセスする場合は、DNS サーバで 2 つの CNAME レコードを作成し、Envoy サービス ロード バランサの `External-IP` アドレスを Grafana ダッシュボードにマッピングします。

Envoy NodePort を使用した Grafana ダッシュボードへのアクセス

NodePort タイプの前提条件である Contour Envoy サービスがデプロイされていて、Grafana 構成ファイル内で指定されている場合は、ワーカー ノードの仮想マシン IP アドレスを取得し、Grafana FQDN の DNS レコードを作成します。

1 クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

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

2 クラスタ内のノードを一覧表示します。

```
kubectl get virtualmachines
```

3 ワーカー ノードの 1 つを選択し、次のコマンドを使用して記述します。

```
kubectl describe virtualmachines tkgs-cluster-X-workers-9twdr-59bc54dc97-kt4cm
```

4 `Vm Ip: 10.115.22.43` などの仮想マシンの IP アドレスを見つけます。

5 Grafana 拡張機能のインストールを確認するには、次の例のように、ワーカー ノードの IP アドレスにマッピングされる Grafana の FQDN を指定してローカルの `/etc/hosts` ファイルを更新します。

```
127.0.0.1 localhost
127.0.1.1 ubuntu
# TKG Grafana with Envoy NodePort
10.115.22.43 grafana.system.tanzu
```

6 `https://grafana.system.tanzu` に移動して、Grafana ダッシュボードにアクセスします。

サイトでは自己署名証明書が使用されるため、ダッシュボードにアクセスする前にブラウザ固有のセキュリティ警告を確認しなければならない場合があります。

grafana-data-values.yaml

次の `grafana-data-values.yaml` ファイルの例を参照してください。

```
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

次の `grafana.yaml` 仕様の例を参照してください。必要に応じて、パッケージのバージョンを更新します。

```
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
```

vSphere 7.x 用の TKr への Harbor のインストール

次の手順に従って、vSphere 7.x 用の TKr を使用してプロビジョニングされた TKG クラスタに Harbor をインストールします。

前提条件

vSphere 7.x の TKr に標準パッケージをインストールするためのワークフローを参照してください。

Harbor には HTTP/S Ingress が必要です。Harbor サービスは Contour パッケージの Envoy サービスを介して公開されます。前提条件として、Contour パッケージをデプロイします。vSphere 7.x 用の TKr への Contour のインストールを参照してください。

- スーパーバイザー に対して NSX ネットワークを使用している場合は、LoadBalancer タイプの Envoy サービスを作成します。
- スーパーバイザー に対して vSphere Distributed Switch ネットワークを使用している場合は、環境と要件に応じて LoadBalancer タイプまたは NodePort タイプの Envoy サービスを作成します。

Harbor 拡張機能には、DNS が必要です。テストと検証のために、Harbor および Notary の FQDN をローカル `/etc/hosts` ファイルに追加します。以下の手順では、これを行う方法を説明します。

本番環境の Harbor では、BIND などのローカル DNS サーバ、または AWS Route53 や Azure DNS などのパブリック クラウドのいずれかの DNS ゾーンが必要です。DNS を設定した後、Harbor の FQDN を DNS サーバに自動的に登録するには、ExternalDNS 拡張機能をインストールします。vSphere 7.x 用の TKr への ExternalDNS のインストールを参照してください。

Harbor のインストール

標準パッケージを使用して Harbor レジストリをインストールするには、次の手順を実行します。

1 リポジトリで使用可能な Harbor のバージョンを一覧表示します。

```
kubectl get packages -n tkg-system | grep harbor
```

2 `harbor.yaml` 仕様を作成します。

```
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 環境に適した値（ホスト名、パスワード、シークレット、ストレージ クラスなど）を使用して、`harbor.yaml` 仕様の `harbor-data-values` シークレットをカスタマイズします。

詳細については、「Harbor パッケージ リファレンス」を参照してください。

4 Harbor をインストールします。

```
kubectl apply -f harbor.yaml
```

5 Harbor のインストールを確認します。

```
kubectl get all -n harbor-registry
```

Envoy LoadBalancer を使用する Harbor 向けの DNS の構成（NSX ネットワーク）

前提条件の Envoy サービスが LoadBalancer を介して公開される場合は、ロード バランサの外部 IP アドレスを取得し、Harbor の FQDN の DNS レコードを作成します。

1 LoadBalancer タイプの Envoy サービスの `External-IP` アドレスを取得します。

```
kubectl get service envoy -n tanzu-system-ingress
```

返された `External-IP` アドレスが次の例のように表示されます。

```
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
```

または、次のコマンドを使用して `External-IP` アドレスを取得することもできます。

```
kubectl get svc envoy -n tanzu-system-ingress -o
jsonpath='{.status.loadBalancer.ingress[0]}'
```

2 Harbor 拡張機能のインストールを確認するには、次の例のように、ロード バランサの `External-IP` アドレスにマッピングされる Harbor および Notary の FQDN を指定してローカルの `/etc/hosts` ファイルを更新します。

```
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 Harbor 拡張機能のインストールを確認するために、Harbor にログインします。

4 Envoy サービスのロード バランサの `External-IP` アドレスを Harbor の FQDN と Notary の FQDN にマッピングする 2 つの CNAME レコードを DNS サーバに作成します。

5 外部 DNS 拡張機能をインストールします。

Envoy NodePort を使用する Harbor 向けの DNS の構成（Distributed Switch ネットワーク）

前提条件の Envoy サービスが NodePort を介して公開される場合は、ワーカー ノードの仮想マシンの IP アドレスを取得し、Harbor の FQDN の DNS レコードを作成します。

注： NodePort を使用するには、`harbor-data-values.yaml` ファイルに正しい `port.https` 値を指定しておく必要があります。

1 クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

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

2 クラスタ内のノードを一覧表示します。

```
kubectl get virtualmachines
```

3 ワーカー ノードの 1 つを選択し、次のコマンドを使用して記述します。

```
kubectl describe virtualmachines tkg2-cluster-X-workers-9twdr-59bc54dc97-kt4cm
```

4 `Vm Ip: 10.115.22.43` などの仮想マシンの IP アドレスを見つけます。

5 Harbor 拡張機能のインストールを確認するには、次の例のように、ワーカー ノードの IP アドレスにマッピングされる Harbor および Notary の FQDN を指定してローカルの `/etc/hosts` ファイルを更新します。

```
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 Harbor 拡張機能のインストールを確認するために、Harbor にログインします。

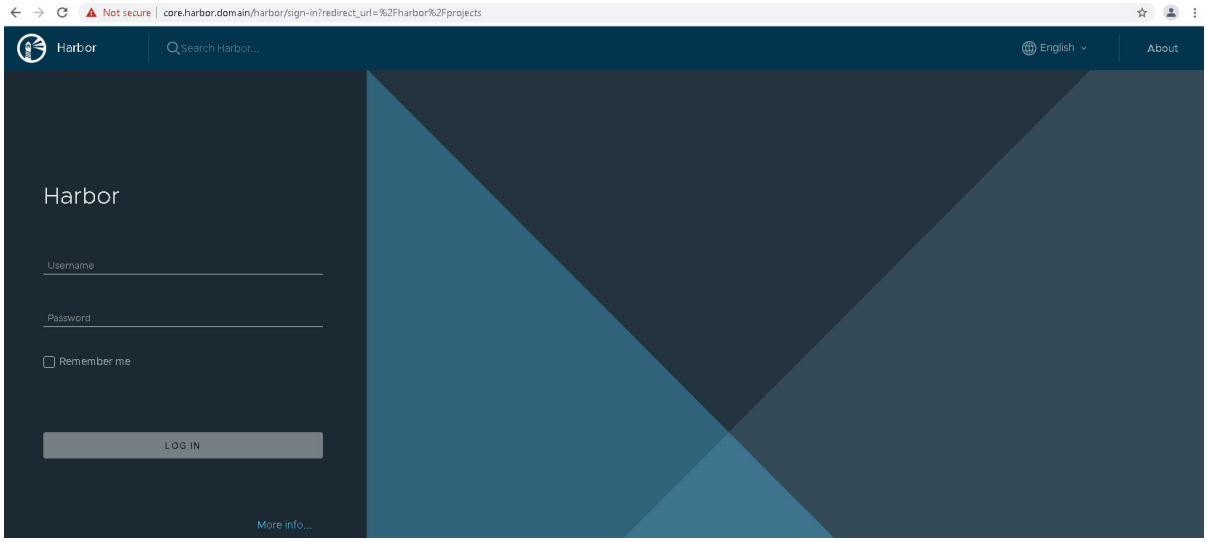
7 ワーカー ノードの IP アドレスを Harbor の FQDN と Notary の FQDN にマッピングする 2 つの CNAME レコードを DNS サーバに作成します。

8 外部 DNS 拡張機能をインストールします。

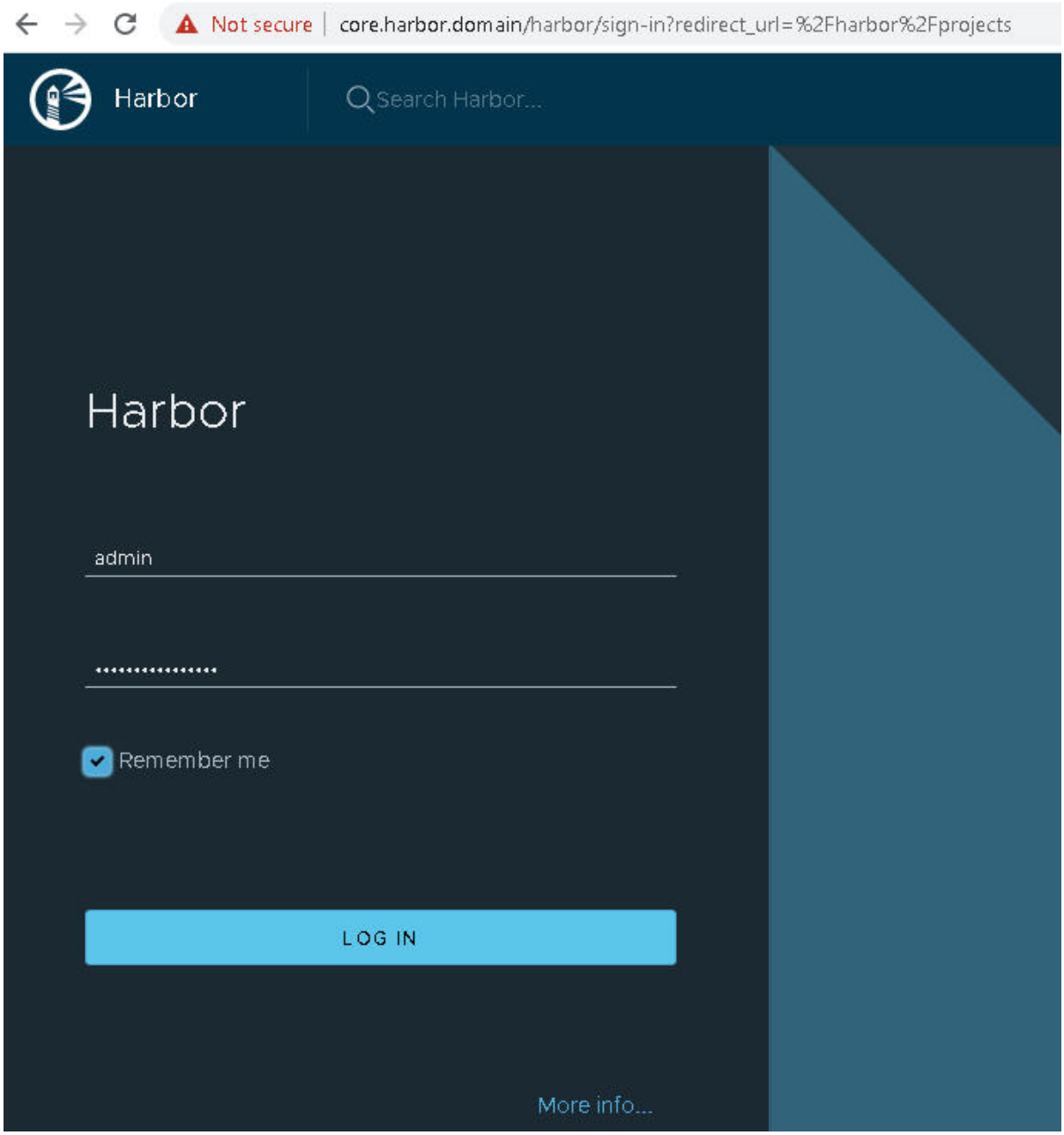
Harbor Web インターフェイスへのログイン

Harbor をインストールして構成したら、ログインして使用を開始します。

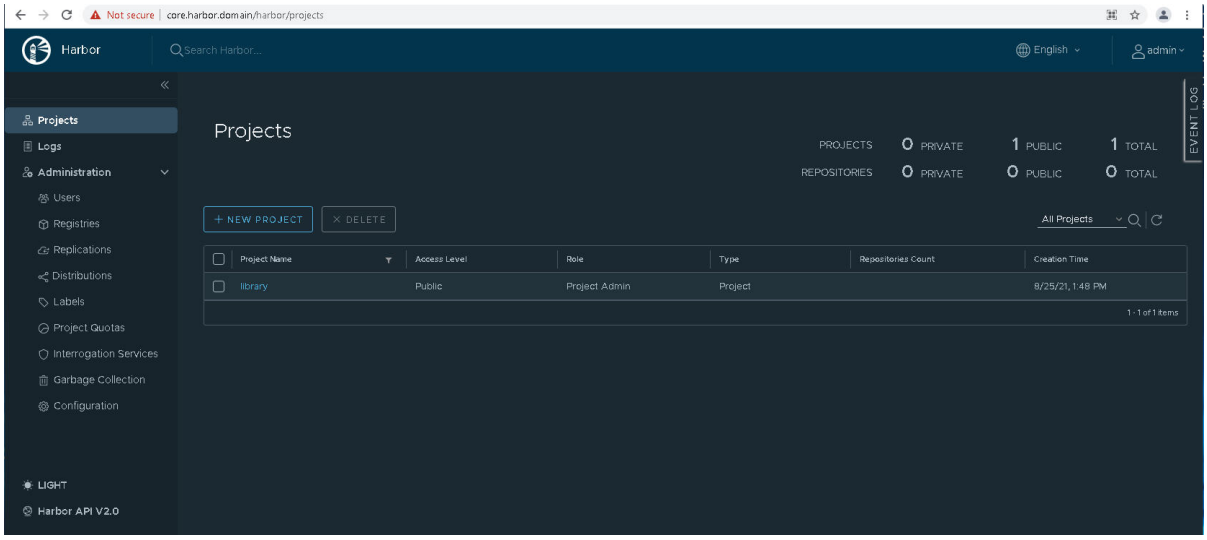
1 Harbor レジストリの Web インターフェイス (`https://core.harbor.domain`)、または使用したホスト名にアクセスします。



2 ユーザー名 **admin** と、生成され、`harbor-data-values.yaml` ファイルに入力したたパスワードを使用して Harbor にログインします。



3 Harbor ユーザー インターフェイスにアクセスできることを確認します。



4 Harbor CA 証明書を取得します。

Harbor インターフェイスで、[プロジェクト] - [ライブラリ] の順に選択するか、[新しいプロジェクト] を作成します。

[レジストリ証明書] をクリックして、Harbor CA 証明書 (`ca.crt`) をダウンロードします。

5 Harbor CA 証明書を Docker クライアントのトラスト ストアに追加して、Harbor レジストリとの間でコンテナ イメージをプッシュおよびプルできるようにします。14 章 TKG サービス クラスタでのプライベート レジストリの使用を参照してください。

6 Harbor の使用方法の詳細については、Harbor のドキュメントを参照してください。

TKG サービス クラスタへのワークロードのデプロイ

12

アプリケーション ワークロードは、ポッド、サービス、パーシステント ボリュームのほか、デプロイやレプリカのセットなどの上位レベルのリソースを使用して TKG サービス クラスタにデプロイできます。

次のトピックを参照してください。

- ロード バランサ サービスを使用したポッドのデプロイ
- 固定 IP アドレスを使用するロード バランサ サービス
- NGINX を使用した Ingress
- Contour を使用した Ingress
- パーシステント ボリュームのストレージ クラスの使用
- パーシステント ストレージ ボリュームの動的な作成
- パーシステント ストレージ ボリュームの静的な作成
- TKG クラスタへの Guestbook アプリケーションのデプロイ
- Guestbook アプリケーションの YAML
- 遅延バインド ボリューム接続を使用した vSphere Zones 全体への StatefulSet アプリケーションのデプロイ

ロード バランサ サービスを使用したポッドのデプロイ

TKG 2.0 クラスタで実行されているポッドに外部トラフィックをルーティングするには、LoadBalancer タイプのサービスを作成します。ロード バランサ サービスはパブリック IP アドレスを公開し、そのアドレスからの受信トラフィックがポッドにルーティングされます。

サービスとして公開される Kubernetes ポッドに外部ロード バランサをプロビジョニングできます。たとえば、NGINX コンテナをデプロイして、LoadBalancer タイプの Kubernetes サービスとして公開できます。

前提条件

- Kubernetes のドキュメントで LoadBalancer サービス タイプについて確認します。
- TKG クラスタをプロビジョニングします。
- TKG クラスタに接続します。

手順

1 次の `nginx-lbsvc.yaml` YAML ファイルを作成します。

この YAML ファイルは、LoadBalancer タイプの Kubernetes サービスを定義し、このサービスの外部ロード バランサとして NGINX コンテナをデプロイします。

```
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 YAML を適用します。

```
kubectl apply -f nginx-lbsvc.yaml
```

3 NGINX サービスが展開されていることを確認します。

```
kubectl get services
```

`srvclb-ngnx` は、外部および内部 IP アドレスで稼動しています。

```
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 ブラウザを使用して、NGINX LoadBalancer サービスの外部 IP アドレスを入力します。

`NGINX` のメッセージ バナーとロード バランサの詳細が表示されます。

固定 IP アドレスを使用するロード バランサ サービス

固定 IP アドレスを使用するように LoadBalancer タイプの Kubernetes サービスを構成できます。この機能を実装する場合は、コンポーネントの最小要件、セキュリティに関する重要な考慮事項、およびクラスタのセキュリティ強化に関するガイダンスに注意してください。

LoadBalancer タイプのサービスにおける固定 IP アドレスの使用

通常、LoadBalancer タイプの Kubernetes サービスを定義すると、ロード バランサによって IP アドレスが一時的に割り当てられます。

ロード バランサの固定 IP アドレスを指定することもできます。サービスの作成時に、割り当てられた固定 IP アドレスを使用してロード バランサのインスタンスがプロビジョニングされます。

次のサービスの例では、固定 IP アドレスを使用してサポート対象のロード バランサを構成する方法を示します。サービス仕様に `loadBalancerIP` パラメータと IP アドレス値を含めます。この例では、IP アドレスは `10.11.12.49` です。

```
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
```

NSX Advanced Load Balancer の場合は、ロード バランサのインストール時に構成された IP アドレス管理プール内の IP アドレスを使用します。サービスが作成され、固定 IP アドレスが割り当てられると、ロード バランサは割り当て済みとマークし、一時的な IP アドレスと同じように IP アドレスのライフサイクルを管理します。つまり、サービスが削除されると、IP アドレスは割り当て解除され、再割り当てが可能になります。

NSX-T ロード バランサの場合は、2 つのオプションがあります。デフォルトのメカニズムは、NSX Advanced Load Balancer と同じように、ロード バランサのインストール時に構成された IP アドレス プールから取得された IP アドレスを使用します。固定 IP アドレスが割り当てられると、ロード バランサは自動的に割り当て済みとマークし、そのライフサイクルを管理します。

NSX-T の 2 番目のオプションでは、固定 IP アドレスを手動で事前に割り当てます。この場合、ロード バランサに割り当てられた外部ロード バランサの IP アドレス プールに含まれない、フローティング IP アドレス プールから取得された IP アドレスを使用します。この場合は、NSX Manager を使用して、IP アドレスの割り当てとライフサイクルを手動で管理します。

セキュリティに関する重要な検討事項とセキュリティ強化要件

この機能を使用する場合は、セキュリティ上の問題が発生する可能性を認識しておく必要があります。開発者が `Service.status.loadBalancerIP` 値にパッチを適用できる場合、その開発者は、パッチが適用された IP アドレス宛てのクラスタ内のトラフィックをハイジャックできる可能性があります。特に、この機能が実装されているクラスタで、`patch` 権限を持つ Role または ClusterRole がサービスまたはユーザー アカウントにバインドされている場合は、そのアカウント所有者が自身の認証情報を使用して `kubectl` コマンドを発行し、ロード バランサに割り当てられた固定 IP アドレスを変更できます。

ロード バランサ サービスに固定 IP アドレスを割り当てた場合に生じる可能性のあるセキュリティ面の影響を回避するには、この機能を実装している各クラスタのセキュリティを強化する必要があります。これを行うには、開発者に対して定義した Role または ClusterRole で、`apiGroups:""` および `resources: services/status` の verb として `patch` を禁止する必要があります。ロール スニペットの例は、この機能を実装する際の禁止項目を示しています。

パッチ適用の禁止

```
- apiGroups:
  - ""
  resources:
  - services/status
  verbs:
  - patch
```

開発者にパッチ権限が付与されているかどうかを確認するには、次のコマンドをそのユーザーとして実行します。

```
kubectl --kubeconfig <KUBECONFIG> auth can-i patch service/status
```

このコマンドによって `yes` が返された場合、そのユーザーにはパッチ権限が付与されています。詳細については、Kubernetes のドキュメントの Checking API Access を参照してください。

開発者にクラスタへのアクセス権を付与する方法については、以下を参照してください。開発者への TKG サービス クラスタへの vCenter SSO アクセス権付与。カスタマイズ可能なサンプルのロール テンプレートの例については、TKG サービス クラスタへのデフォルトのポッド セキュリティ ポリシーの適用を参照してください。クラスタへのアクセスを制限する方法の例については、https://kubernetes.io/docs/reference/access-authn-authz/rbac/#role-example を参照してください。

NGINX を使用した Ingress

Kubernetes Ingress のリソースは、クラスタ外からクラスタ内の 1 つ以上のサービスに HTTP または HTTPS ルーティングを提供します。TKG クラスタは、NGINX などのサードパーティ製コントローラを介した入力をサポートします。

このチュートリアルは、Tanzu Kubernetes クラスタ内のサービスに外部トラフィックをルーティングする際に、NGINX に基づく Kubernetes Ingress サービスをデプロイする方法を示しています。Ingress サービスには Ingress コントローラが必要です。NGINX Ingress Controller は、Helm を使用してインストールします。Helm は Kubernetes のパッケージ マネージャです。

注： このタスクを実行する方法はいくつかあります。ここに記載されている手順では、1 つの方法を示します。環境によっては、他の方法が適している場合があります。

前提条件

- Kubernetes のドキュメントで、Ingress のリソースについて確認します。
- NGINX Ingress Controller のドキュメントを確認してください。
- TKG クラスタをプロビジョニングします。
- 必要に応じてポッド セキュリティ ポリシーを有効にします。
- TKG クラスタに接続します。

手順

1 Helm ドキュメントを参照して、Helm をインストールします。

2 Helm を使用して NGINX Ingress Controller をインストールします。

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm install ingress-nginx ingress-nginx/ingress-nginx
```

3 NGINX Ingress Controller が LoadBalancer タイプのサービスとしてデプロイされていることを確認します。

```
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 外部 IP アドレスを使用してロード バランサを ping します。

```
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 NGINX Ingress Controller が実行されていることを確認します。

```
kubectl get pods
```

```
NAME                                        READY   STATUS    RESTARTS   AGE
ingress-nginx-controller-7c6c46898c-v6blt   1/1     Running   0          76m
```

6 Ingress ルールと `ingress-hello.yaml` という名前のパスを含む Ingress リソースを作成します。

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-hello
spec:
  rules:
  - http:
      paths:
      - path: /hello
        backend:
          serviceName: hello
          servicePort: 80
```

注目: Kubernetes API `networking.k8s.io/v1beta1` は、Kubernetes v1.22 以降では廃止されました。新しく使用される API は `networking.k8s.io/v1` です。https://kubernetes.io/docs/reference/using-api/deprecation-guide/ に記載されているように、いくつかのマニフェストが変更されています。

7 `ingress-hello` リソースをデプロイします。

```
kubectl apply -f ingress-hello.yaml
```

```
ingress.networking.k8s.io/ingress-hello created
```

8 Ingress リソースがデプロイされたことを確認します。

IP アドレスは Ingress コントローラの外部 IP アドレスにマッピングされることに注意してください。

```
kubectl get ingress
```

```
NAME            CLASS    HOSTS   ADDRESS         PORTS   AGE
ingress-hello   <none>   *       10.19.14.76     80      51m
```

9 hello テスト アプリケーションおよび `ingress-hello-test.yaml` という名前のサービスを作成します。

```
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 `ingress-hello-test` リソースをデプロイします。

```
kubectl apply -f ingress-hello-test.yaml
```

```
service/hello created
deployment.apps/hello created
```

11 `hello` デプロイを使用できることを確認します。

```
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 NGINX Ingress Controller によって使用されるロード バランサのパブリック IP アドレスを取得します。

```
kubectl get ingress
```

```
NAME            CLASS    HOSTS   ADDRESS         PORTS   AGE
ingress-hello   <none>   *       10.19.14.76     80      13m
```

13 ブラウザを使用してパブリック IP アドレスに移動し、Ingress のパスを含めます。

```
http://10.19.14.76/hello
```

「hello」というメッセージが返されます。

```
{"message":"Hello"}
```

結果

クラスタ内で実行されているサービスの背後にあるバックエンド アプリケーションに外部からアクセスするには、ロード バランサの外部 IP アドレスを使用して、Ingress コントローラを介してブラウザからアクセスします。

Contour を使用した Ingress

Kubernetes Ingress のリソースは、クラスタ外からクラスタ内の 1 つ以上のサービスに HTTP または HTTPS ルーティングを提供します。TKG クラスタは、Contour などのサードパーティ製コントローラを介した入力をサポートします。

このチュートリアルは、TKG クラスタ内のサービスに外部トラフィックをルーティングするための、Contour 入力方向コントローラをデプロイする方法を示しています。Contour は、VMware が参加しているオープンソース プロジェクトです。

前提条件

- Kubernetes のドキュメントで、Ingress のリソースについて確認します。
- Contour Ingress コントローラを確認します。
- TKG クラスタをプロビジョニングします。
- TKG クラスタに接続します。

手順

1 ClusterRoleBinding を作成し、サービス アカウントでクラスタのすべてのリソースを管理できるようにします。

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding
--clusterrole=psp:vmware-system-privileged --group=system:authenticated
```

注： セキュリティを強化する必要がある場合は、`projectcontour` 名前空間で RoleBinding を使用します。TKG サービス クラスタへのデフォルトのポッド セキュリティ ポリシーの適用を参照してください。

2 `projectcontour` という名前の名前空間を作成します。

これが、Contour 入力方向コントローラのデプロイで使用するデフォルトの名前空間になります。

```
kubectl create ns projectcontour
```

3 Contour 入力方向コントローラの最新の YAML である Contour Ingress Deployment をダウンロードします。

4 テキスト エディタで `contour.yaml` ファイルを開きます。

5 次の 2 つの行の前に `#` 記号を付加して、各行をコメントアウトします。

行 1632：

```
# service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp
```

行 1634：

```
# externalTrafficPolicy: Local
```

6 `contour.yaml` ファイルを適用して Contour をデプロイします。

```
kubectl apply -f contour.yaml
```

7 Contour 入力方向コントローラと Envoy ロード バランサ サービスがデプロイされていることを確認します。

```
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 Contour および Envoy ポッドが実行中であることを確認します。

```
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 外部 IP アドレスを使用してロード バランサを ping します。

```
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 `ingress-nihao.yaml` という名前の入力方向リソースを作成します。

YAML を作成します。

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-nihao
spec:
  rules:
  - http:
      paths:
      - path: /nihao
        backend:
          serviceName: nihao
          servicePort: 80
```

YAML を適用します。

```
kubectl apply -f ingress-nihao.yaml
```

入力方向リソースが作成されたことを確認します。

```
kubectl get ingress
```

入力方向オブジェクトでは、Envoy LoadBalancer の外部 IP アドレス（この例では `192.168.123.5`）が使用されます。

```
NAME            CLASS    HOSTS   ADDRESS         PORTS   AGE
ingress-nihao   <none>   *       192.168.123.5     80      17s
```

11 バックエンド アプリケーションを使用してテスト サービスをデプロイします。

`ingress-nihao-test.yaml` という名前の YAML ファイルを作成します。

```
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
```

YAML を適用します。

```
kubectl apply -f ingress-nihao-test.yaml
```

`nihao` サービスが作成されたことを確認します。

```
kubectl get services
```

```
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
nihao        ClusterIP   10.14.21.22     <none>        80/TCP     15s
```

バックエンド デプロイが作成されたことを確認します。

```
kubectl get deployments
```

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nihao   3/3     3            3           2m25s
```

バックエンド ポッドが存在することを確認します。

```
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 Contour 入力方向コントローラで使用されるロード バランサのパブリック IP アドレスを取得します。

```
kubectl get ingress
```

```
NAME             CLASS    HOSTS   ADDRESS          PORTS   AGE
ingress-nihao    <none>   *       10.19.14.76      80      13m
```

13 ブラウザを使用してパブリック IP アドレスに移動し、Ingress のパスを含めます。

```
http://10.19.14.76/nihao
```

「hello」というメッセージが返されます。

```
{"message":"Hello"}
```

結果

クラスタ内で実行されているサービスの背後にあるバックエンド アプリケーションに外部からアクセスするには、ロード バランサの外部 IP アドレスを使用して、Ingress コントローラを介してブラウザからアクセスします。

パーシステント ボリュームのストレージ クラスの使用

vSphere 名前空間 に割り当てられた vSphere ストレージ ポリシーにより、パーシステント ボリュームで使用可能なストレージ クラスのエディションが 2 つ生成されます。選択するエディションは、要件によって異なります。

使用可能なストレージ クラスの 2 つのエディション

vSphere ストレージ ポリシーを vSphere 名前空間 に割り当てると、一致する Kubernetes ストレージ クラスがその vSphere 名前空間 に作成されます。このストレージ クラスは、その名前空間でプロビジョニングされた各 TKG クラスタにレプリケートされます。その後、このストレージ クラスを使用して、クラスタ ワークロードに使用するパーシステント ストレージ ボリュームを作成できます。

vSphere 名前空間 に割り当てられた各 vSphere ストレージ ポリシーごとに、「Immediate」バインド モードのストレージ クラスが スーパーバイザー に 1 つ存在します。

```
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>
```

その vSphere 名前空間 で展開される TKG クラスタごとに、2 つのストレージ クラスがあります。1 つは、スーパーバイザー 上の対応するストレージ クラスと同じものです。もう 1 つは、名前に `*-latebinding` が付加され、バインド モードが「WaitForFirstConsumer」になっています。

```
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
```

Kubernetes スケジューラによってコンピューティングが選択された後でパーシステント ボリュームをプロビジョニングする場合は、`latebinding` バージョンのストレージ クラスを使用します。詳細については、Kubernetes ドキュメントの Volume Binding Mode を参照してください。

```
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>
```

ストレージ クラスへのパッチ適用

スーパーバイザー の TKG の場合、kubectl と YAML を使用してストレージ クラスを手動で作成することはできません。vSphere ストレージ ポリシー フレームワークを使用してストレージ クラスを作成し、vSphere 名前空間に適用することのみが可能です。その結果、この vSphere 名前空間 でプロビジョニングされた TKG クラスタごとに 2 つの対応するストレージ クラスが作成されます。

kubectl と YAML を使用してストレージ クラスを手動で作成することはできませんが、kubectl を使用して既存のストレージ クラスを変更することはできます。この操作が必要になるのは、デフォルトのストレージ クラスを指定せずに TKG クラスタをプロビジョニングした後に、デフォルトのストレージ クラスを必要とする Helm または Tanzu パッケージを使用してアプリケーションをデプロイする必要が生じた場合です。

デフォルトのストレージを使用してまったく新しいクラスタを作成する代わりに、Kubernetes のドキュメントデフォルトの StorageClass の変更で説明されているように既存のストレージ クラスにパッチを適用して、default = true の注釈を追加することができます。

たとえば、次のコマンドは、TKG クラスタで使用可能な 2 つのストレージ クラスを返します。

```
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>
```

次のコマンドを使用して、ストレージ クラスの 1 つにパッチを適用し、注釈を追加します。

```
kubectl patch storageclass gc-storage-profile -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class":"true"}}}'
storageclass.storage.k8s.io/gc-storage-profile patched
```

ストレージ クラスを再度確認すると、ストレージ クラスの 1 つにパッチが適用され、デフォルトになっていることがわかります。

```
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>
```

パーシステント ストレージ ボリュームの動的な作成

既存のストレージ クラスとパーシステント ボリューム要求 (PVC) を使用して、パーシステント ストレージ ボリュームを動的に作成できます。

TKG クラスタの動的 PVC

TKG クラスタでステートフル ワークロードを実行するために、パーシステント ボリューム要求 (PVC) を作成して、基盤となるストレージ インフラストラクチャの詳細を把握していなくてもパーシステント ストレージ リソースを要求することができます。PVC に使用されるストレージは、vSphere 名前空間 用のストレージ割り当てから割り当てられます。

この要求により、パーシステント ボリューム オブジェクトとそれに対応する仮想ディスクが動的にプロビジョニングされます。要求はパーシステント ボリュームにバインドされています。この要求を削除すると、対応するパーシステント ボリューム オブジェクトおよびプロビジョニングされた仮想ディスクも削除されます。

PVC を作成すると、バッキング パーシステント ボリュームが動的に作成されます。PVC は、**tkg-store** ストレージ クラスを参照します。ストレージ クラスは、ターゲット TKG クラスタがプロビジョニングされている vSphere 名前空間 に関連付けられています。詳細についてはパーシステント ボリュームのストレージ クラスの使用を参照してください。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: tkg-cluster-pvc
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: tkg-store
  resources:
    requests:
        storage: 3Gi
```

PVC を作成します。

```
kubectl apply -f pvc_name.yaml
```

PVC を確認します。

```
kubectl get pvc my-pvc
```

ポッドまたはデプロイ仕様で PVC を指定します。例：

```
...
volumes:
    - name: my-pvc
      persistentVolumeClaim:
        claimName: my-pvc
```

パーシステント ストレージ ボリュームの静的な作成

スーパーバイザー からパーシステント ボリューム要求 (PVC) を使用して、TKG 2.0 クラスタ内にパーシステント ボリューム (PV) を静的に作成することができます。

パーシステント ボリュームの定義

次に、静的パーシステント ボリューム (PV) の定義の例を示します。定義には、ストレージ クラスとボリューム ハンドルが必要です。`volumeHandle` は、ターゲット TKG クラスタがプロビジョニングされているものと同じ vSphere 名前空間 の スーパーバイザー に作成されたパーシステント ボリューム要求 (PVC) の名前です。この PVC は、どのポッドにも接続しません。

`storageClassName` を取得するには、次のコマンドを使用します。

```
kubectl get storageclass
```

`volumeHandle` で、スーパーバイザー での PVC の名前を入力します。

```
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.
```

次の手順を使用して PV を作成します。

```
kubectl apply -f redis-leader-pvc.yaml -n guestbook
```

静的に定義された PV に対するパーシステント ボリューム要求 (PVC)

複数のゾーンにまたがって スーパーバイザー をデプロイした場合。

`storageClassName` を PV と同じ値に設定します。

```
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
```

作成した PV に PVC がバインドされていることを確認します。

```
kubectl get pv,pvc
```

TKG クラスタへの Guestbook アプリケーションのデプロイ

Guestbook アプリケーションを TKG クラスタにデプロイし、Kubernetes の詳細を把握します。

Guestbook アプリケーションをデプロイすることは、Kubernetes の詳細を把握するために役立つ手段です。アプリケーションは、Deployment オブジェクトと ReplicaSet オブジェクトを使用してデフォルトの名前空間にポッドをデプロイし、サービスを使用してこれらのポッドを公開します。Guestbook データは静的であるため、アプリケーションが停止してもデータは残ります。このチュートリアルでは、動的なパーシステント ボリューム要求 (PVC) を使用して、パーシステント ストレージ リソースを要求します。その際、基盤となるストレージの詳細を把握する必要はありません。PVC に使用されるストレージは、vSphere 名前空間 用のストレージ割り当てから割り当てられます。コンテナはデフォルトで短期、かつステートレスです。ステートフル ワークロードの場合、一般的な方法は、パーシステント ボリューム要求 (PVC) を作成することです。PVC を使用してパーシステント ボリュームをマウントし、ストレージにアクセスできます。この要求により、パーシステント ボリューム オブジェクトとそれに対応する仮想ディスクが動的にプロビジョニングされます。要求はパーシステント ボリュームにバインドされています。この要求を削除すると、対応するパーシステント ボリューム オブジェクトおよびプロビジョニングされた仮想ディスクも削除されます。

前提条件

次のトピックを参照してください。

- Kubernetes ドキュメントの「Guestbook アプリケーションのチュートリアル」
- TKG クラスタをプロビジョニングします。
- TKG クラスタに接続します。

手順

1 TKG クラスタにログインします。

2 Guestbook 名前空間を作成します。

```
kubectl create namespace guestbook
```

以下を確認します。

```
kubectl get ns
```

3 デフォルトの特権 PSP を使用して、ロールベースのアクセス制御を作成します。

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding --
clusterrole=psp:vmware-system-privileged --group=system:authenticated
```

注： セキュリティを強化する必要がある場合は、Guestbook 名前空間に RoleBinding を適用します。

4 ストレージ クラスを検証するか、作成します。

既存のストレージ クラスを検証するには、次の手順を実行します。

```
kubectl describe namespace
```

または、vSphere 管理者権限がある場合は、次の手順を実行します。

```
kubectl get storageclass
```

5 ストレージ クラスを参照する動的パーシステント ボリューム要求 (PVC) の YAML ファイルを作成します。

次の YAML ファイルを使用します。各ファイルをストレージ クラスの名前で更新します。

- Redis リーダー パーシステント ボリュームの要求
- Redis フォロワー パーシステント ボリュームの要求

6 クラスタに Guestbook PVC を適用します。

```
kubectl apply -f redis-leader-pvc.yaml -n guestbook
```

```
kubectl apply -f redis-follower-pvc.yaml -n guestbook
```

7 PVC のステータスを確認します。

```
kubectl get pvc,pv -n guestbook
```

PVC とパーシステント ボリューム (PV) が一覧表示され、使用できるようになります。

```
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 Guestbook YAML ファイルを作成します。

次の YAML ファイルを使用します。

- Redis リーダーのデプロイ
- Redis フォロワーのデプロイ
- Redis リーダーのサービス
- Redis フォロワー サービス
- Guestbook フロントエンドのデプロイ
- Guestbook フロントエンド サービス

9 名前空間に Guestbook アプリケーションをデプロイします。

```
kubectl apply -f . --namespace guestbook
```

10 Guestbook リソースの作成を確認します。

```
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 `service/guestbook-frontend` ロード バランサの `External-IP` アドレス (この例では `10.19.15.99`) を使用して Guestbook Web ページにアクセスします。

Guestbook Web インターフェイスが表示され、Guestbook データベースに値を入力できます。アプリケーションを再起動すると、データは保持されます。

Guestbook アプリケーションの YAML

サンプル YAML ファイルを使用して、パーシステント データ ストレージとともに Guestbook アプリケーションをデプロイします。

Redis リーダー パーシステント ボリュームの要求

ファイル `redis-leader-pvc.yaml` は、指定したストレージ クラスを参照するパーシステント ボリュームの要求の例です。この例を使用するには、ストレージ クラスの名前を入力します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: redis-leader-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: tkg-storage-class-name
  resources:
    requests:
      storage: 2Gi
```

Redis フォロワー パーシステント ボリュームの要求

ファイル `redis-follower-pvc.yaml` は、指定したストレージ クラスを参照するパーシステント ボリュームの要求の例です。この例を使用するには、ストレージ クラスの名前を入力します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: redis-follower-pvc
spec:
```

```
  accessModes:
    - ReadWriteOnce
  storageClassName: tkg-storage-class-name
  resources:
    requests:
      storage: 2Gi
```

Redis リーダーのデプロイ

ファイル `redis-leader-deployment.yaml` は、パーシステント ボリュームを使用した Redis リーダーのデプロイ例です。

```
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 フォロワーのデプロイ

ファイル `redis-follower-deployment.yaml` は、パーシステント ボリュームを使用した Redis フォロワーのデプロイ例です。

```
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 リーダーのサービス

ファイル `redis-leader-service.yaml` は、Redis リーダーのサービス例です。

```
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 フォロワー サービス

ファイル `redis-follower-service.yaml` は、Redis フォロワーのサービス例です。

```
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 フロントエンドのデプロイ

ファイル `guestbook-frontend-deployment.yaml` は、Guestbook フロントエンドのデプロイ例です。

```
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 フロントエンド サービス

ファイル `guestbook-frontend-service.yaml` は、Guestbook フロントエンド ロード バランサ サービスの例です。

```
apiVersion: v1
kind: Service
metadata:
  name: guestbook-frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend
```

遅延バインド ボリューム接続を使用した vSphere Zones 全体への StatefulSet アプリケーションのデプロイ

この例では、StatefulSet アプリケーションを スーパーバイザー 上の TKG クラスタにデプロイする方法を示します。

ストレージ クラス

ストレージ クラスは、2 つのエディションから選択できます。このデプロイでは、`*-latebinding` エディションを使用します。

```
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
```

マルチゾーン スーパーバイザー トポロジ

TKG クラスタは、vSphere Zones 全体の スーパーバイザー にプロビジョニングされます。

```
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

StatefulSet (`sts.yaml`) では、アプリケーションがポッドにデプロイされます。各ポッドには固定的な ID があり、再スケジュールしても維持されます。

```
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
```

アプリケーションのデプロイ

StatefulSet アプリケーションを次のようにデプロイします。デプロイが成功すると、アプリケーション ポッドは遅延バインド ボリューム モードのストレージ クラスを使用する最初のコンシューマの待機を使用して、3 つの vSphere Zones にわたってスケジュール設定されます。

1 StatefulSet をデプロイします。

```
kubectl create -f sts.yaml
statefulset.apps/web created
```

2 StatefulSet を確認します。

```
kubectl get statefulset
NAME   READY   AGE
web    3/3     112s
```

3 ポッドを確認します。

```
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 ゾーンおよび遅延ボリューム バインドに関するポッドのスケジュール設定を確認します。

```
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"]}]}]}}
```

TKG サービス クラスタへの AI/ML ワークロードのデプロイ

13

NVIDIA vGPU を使用して、TKG サービス クラスタに AI/ML ワークロードをデプロイできます。AI/ML ワークロードをデプロイするには、vSphere 管理者がいくつかの初期設定を行い、クラスタ オペレータがいくつかの構成を行う必要があります。環境が vGPU 対応になると、開発者はコンテナベースの AI/ML ワークロードを TKG サービス クラスタにデプロイできます。

次のトピックを参照してください。

- TKG サービス クラスタへの AI/ML ワークロードのデプロイについて
- TKGS クラスタへの AI/ML ワークロードのデプロイに関する vSphere 管理者ワークフロー
- TKG サービス クラスタへの AI/ML ワークロードのデプロイに関するクラスタ オペレータのワークフロー
- NVIDIA vGPU デバイス用のカスタム仮想マシン クラスの作成

TKG サービス クラスタへの AI/ML ワークロードのデプロイについて

NVIDIA GPU テクノロジーを使用して、TKG サービス クラスタに AI/ML ワークロードをデプロイできます。

AI/ML ワークロードの TKGS サポート

コンピューティング集約型のワークロードを TKG サービス クラスタにデプロイできます。このコンテキストでは、計算を多用するワークロードとは、GPU アクセラレータ デバイスを使用する必要がある人工知能 (AI) または機械学習 (ML) アプリケーションを意味します。

Kubernetes 環境で AI/ML ワークロードの実行を容易にするために、VMware は NVIDIA と連携して、NVIDIA GPU Cloud プラットフォームをサポートしています。つまり、NGC カタログ内のコンテナ イメージを TKGS クラスタにデプロイできます。vSphere 8 の NVIDIA GPU サポートの詳細については、Tech Zone の vGPU の記事を参照してください。

サポート対象の GPU モード

TKG サービス クラスタに NVIDIA ベースの AI/ML ワークロードをデプロイするには、Ubuntu エディションの Tanzu Kubernetes リリース（バージョン 1.22 以降）を使用する必要があります。vSphere では 2 つのモード（NVIDIA GRID vGPU、動的 DirectPath I/O デバイスを使用する GPU パススルー）がサポートされます。詳細については、NVIDIA ドキュメントの「Supported Operating Systems and Kubernetes Platforms」を参照してください。

表 13-1. NVIDIA vGPU を使用する vSphere 仮想マシン

OS	TKr	vSphere with Tanzu	説明
Ubuntu 20.04 LTS	1.22 ～ 1.2x*（最新バージョンから 1.28 まで）	7.0 U3c 8.0 U2+	GPU デバイスは、各 ESXi ホストにインストールされている NVIDIA ホスト マネージャ ドライバによって仮想化されます。仮想化された GPU デバイスは、複数の NVIDIA 仮想 GPU (vGPU) 間で共有されます。 **注：** vSphere Distributed Resource Scheduler (DRS) は、vSphere クラスタを構成するホスト全体に幅優先方式で vGPU 仮想マシンを分散します。詳細については、『vSphere リソース管理』ガイドの「vGPU 仮想マシンの DRS 配置」を参照してください。 各 NVIDIA vGPU は、GPU デバイスのメモリ容量によって定義されます。たとえば、GPU デバイスの RAM の合計容量が 32 GB の場合は、それぞれ 4 GB のメモリを持つ 8 つの vGPU を作成できます。

表 13-2. GPU パススルーを使用する vSphere 仮想マシン

OS	TKr	vSphere with Tanzu	説明
Ubuntu 20.04 LTS	1.22 ～ 1.2x*（最新バージョンから 1.28 まで）	7.0 U3c 8.0 U2+	NVIDIA vGPU プロファイルを構成した仮想マシン クラスに、動的 DirectPath IO を使用するパススルー ネットワーク デバイスのサポートを含めます。この場合、仮想マシンの配置は vSphere DRS によって決まります。

TKGS クラスタへの AI/ML ワークロードのデプロイに関する vSphere 管理者ワークフロー

開発者が TKG クラスタに AI/ML ワークロードをデプロイできるようにするには、vSphere 管理者が NVIDIA GPU ハードウェアをサポートするように スーパーバイザー 環境を設定します。

管理者の手順 1：システム要件を確認する

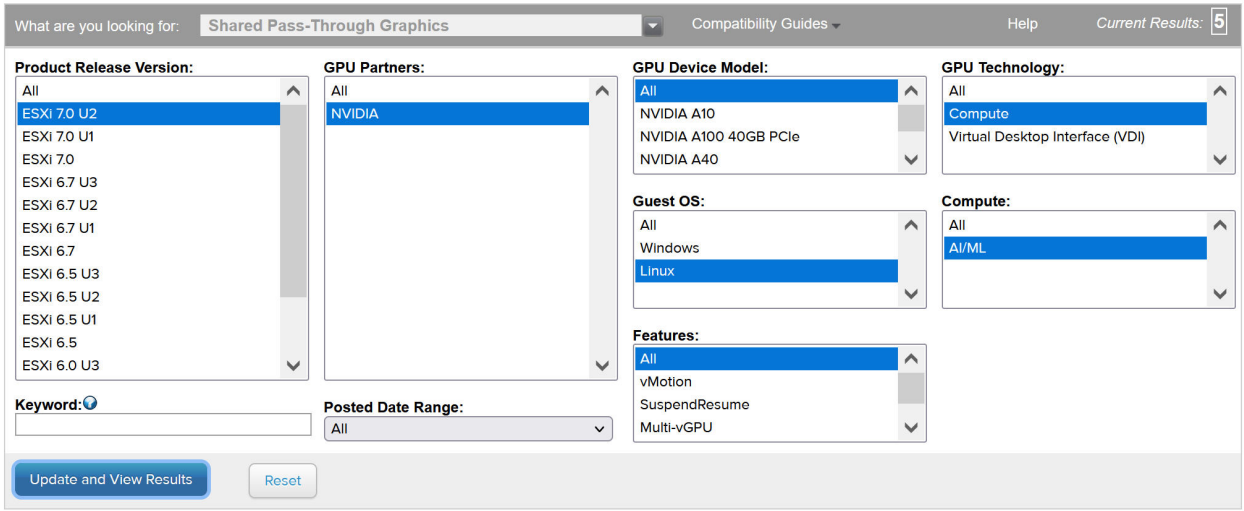
次のシステム要件を参照して、TKG クラスタに AI/ML ワークロードをデプロイできるように環境を設定します。

要件	説明
vSphere 8 インフラストラクチャ	vCenter Server ホストおよび ESXi ホスト
ワークロード管理ライセンス	vSphere 名前空間と スーパーバイザー
TKR Ubuntu OVA	Tanzu Kubernetes リリースのリリース ノート
NVIDIA vGPU ホスト ドライバ	NGC Web サイトから VIB をダウンロードします。詳細については、vGPU ソフトウェア ドライバのドキュメントを参照してください。
vGPU の NVIDIA ライセンス サーバ	組織から提供された FQDN

管理者の手順 2：サポートされている NVIDIA GPU デバイスを ESXi ホストにインストールする

TKG に AI/ML ワークロードをデプロイするには、[ワークロード管理] を有効にする vCenter Server クラスタ内の各 ESXi ホストに、サポートされている NVIDIA GPU デバイスを 1 つ以上インストールします。

互換性のある NVIDIA GPU デバイスを表示するには、VMware 互換性ガイドを参照してください。



Click here to **Read** Important Support Information

Click on the 'GPU Device Model' to view more details and to subscribe to RSS feeds.

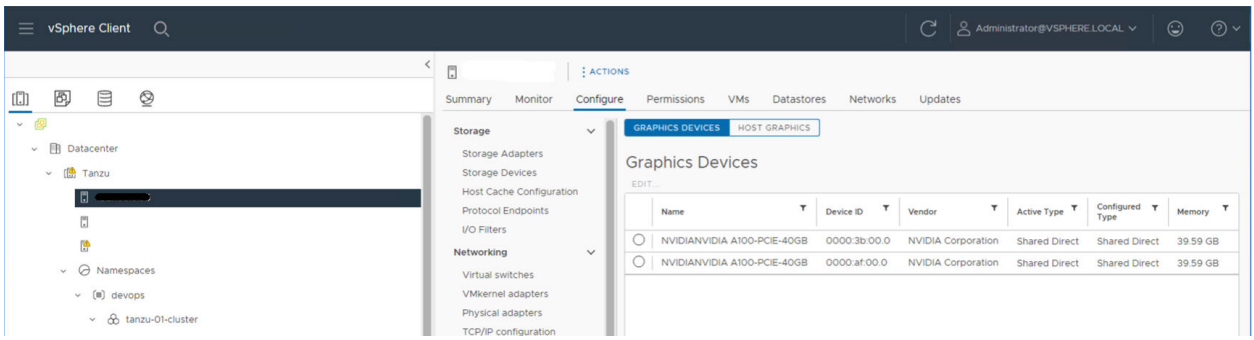
Bookmark | Print | Export to CSV

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

NVIDA GPU デバイスは、最新の NVIDIA AI Enterprise (NVAIE) vGPU プロファイルをサポートしている必要があります。ガイダンスについては、NVIDIA Virtual GPU Software Supported GPUs ドキュメントを参照してください。

たとえば、次の ESXi ホストには、2 つの NVIDIA GPU A100 デバイスがインストールされています。



管理者の手順 3：各 ESXi ホストを vGPU 操作用に構成する

[ワークロード管理] が有効になっている vCenter Server クラスタ内の ESXi ホストごとに、[直接共有] および [SR-IOV] を有効にして、ホストを NVIDIA vGPU 用に構成します。

[各 ESXi ホストでの直接共有の有効化]

NVIDIA vGPU 機能をロック解除するには、[ワークロード管理] が有効になっている vCenter Server クラスタ内の各 ESXi ホストで [直接共有] モードを有効にします。

[直接共有] を有効にするには、次の手順を実行します。その他のガイダンスについては、「vSphere での仮想グラフィックの構成」を参照してください。

1 vSphere Client を使用して、vCenter Server にログインします。

2 vCenter Server クラスタで ESXi ホストを選択します。

3 [構成] - [ハードウェア] - [グラフィック] - [グラフィック デバイス] の順に選択します。

4 NVIDIA GPU アクセラレータ デバイスを選択します。

5 グラフィック デバイスの設定を [編集] します。

6 [直接共有] を選択します。

7 [共有パススルー GPU 割り当てポリシー] で、最適なパフォーマンスを実現するために、[複数の GPU にわたって仮想マシンを分散] を選択します。

8 [OK] をクリックして構成を保存します。

9 ホストの再起動後に設定が有効になります。

10 ESXi ホストを右クリックして、メンテナンス モードにします。

11 ホストを再起動します。

12 ホストが再実行されているときに、メンテナンス モードを終了します。

13 [ワークロード管理] をサポートする vSphere クラスタ内の ESXi ホストごとにこのプロセスを繰り返します。

[NVIDIA GPU A30 デバイスおよび A100 デバイスの SR-IOV BIOS の有効化]

マルチインスタンス GPU（MIG モード）で必要となる NVIDIA A30 デバイスまたは A100 GPU デバイスを使用している場合は、ESXi ホストで SR-IOV を有効にする必要があります。SR-IOV が有効になっていない場合は、Tanzu Kubernetes クラスタ ノード仮想マシンを起動できません。この問題が発生すると、[ワークロード管理] が有効になっている vCenter Server の [最近のタスク] ペインに次のエラー メッセージが表示されます。

```
Could not initialize plugin libnvidia-vgx.so for vGPU nvidia_aXXX-xx. Failed to start the virtual machine. Module DevicePowerOn power on failed.
```

SR-IOV を有効にするには、Web コンソールを使用して ESXi ホストにログインします。[管理] - [ハードウェア] の順に選択します。NVIDIA GPU デバイスを選択して、[SR-IOV の構成] をクリックします。ここで SR-IOV をオンにできます。その他のガイダンスについては、vSphere ドキュメントの Single Root I/O Virtualization (SR-IOV)を参照してください。

[vGPU と動的 DirectPath I/O（パススルー対応デバイス）]

vGPU と動的 DirectPath I/O を使用している場合、以下の追加構成を実行します。

1 vSphere Client を使用して、vCenter Server にログインします。

2 vCenter Server クラスタでターゲット ESXi ホストを選択します。

3 [構成] - [ハードウェア] - [PCI デバイス] の順に選択します。

4 [すべての PCI デバイス] タブを選択します。

5 ターゲット NVIDIA GPU アクセラレータ デバイスを選択します。

6 [パススルーの切り替え] をクリックします。

7 ESXi ホストを右クリックして、メンテナンス モードにします。

8 ホストを再起動します。

9 ホストが再実行されているときに、メンテナンス モードを終了します。

管理者の手順 4：各 ESXi ホストに NVIDIA ホスト マネージャ ドライバをインストールする

NVIDIA vGPU グラフィック アクセラレーションを使用して Tanzu Kubernetes クラスタ ノード仮想マシンを実行するには、[ワークロード管理] を有効にする vCenter Server クラスタ内の各 ESXi ホストに NVIDIA ホスト マネージャ ドライバをインストールします。

NVIDIA vGPU ホスト マネージャ ドライバのコンポーネントは、vSphere インストール バンドル (VIB) に含まれています。NVAIE VIB は、NVIDIA GRID ライセンス プログラムを通じて組織から提供されます。VMware は NVAIE VIB を提供することも、ダウンロード可能にすることもしません。NVIDIA ライセンス プログラムの一環として、ユーザーの組織がライセンス サーバを設定します。詳細については、NVIDIA 仮想 GPU ソフトウェア クイック スタート ガイドを参照してください。

NVIDIA 環境が設定されたら、各 ESXi ホストで次のコマンドを実行して、NVIDIA ライセンス サーバのアドレスと NVAIE VIB のバージョンを環境に適した値に置き換えます。その他のガイダンスについては、VMware サポートのナレッジベースの記事 Installing and configuring the NVIDIA VIB on ESXi を参照してください。

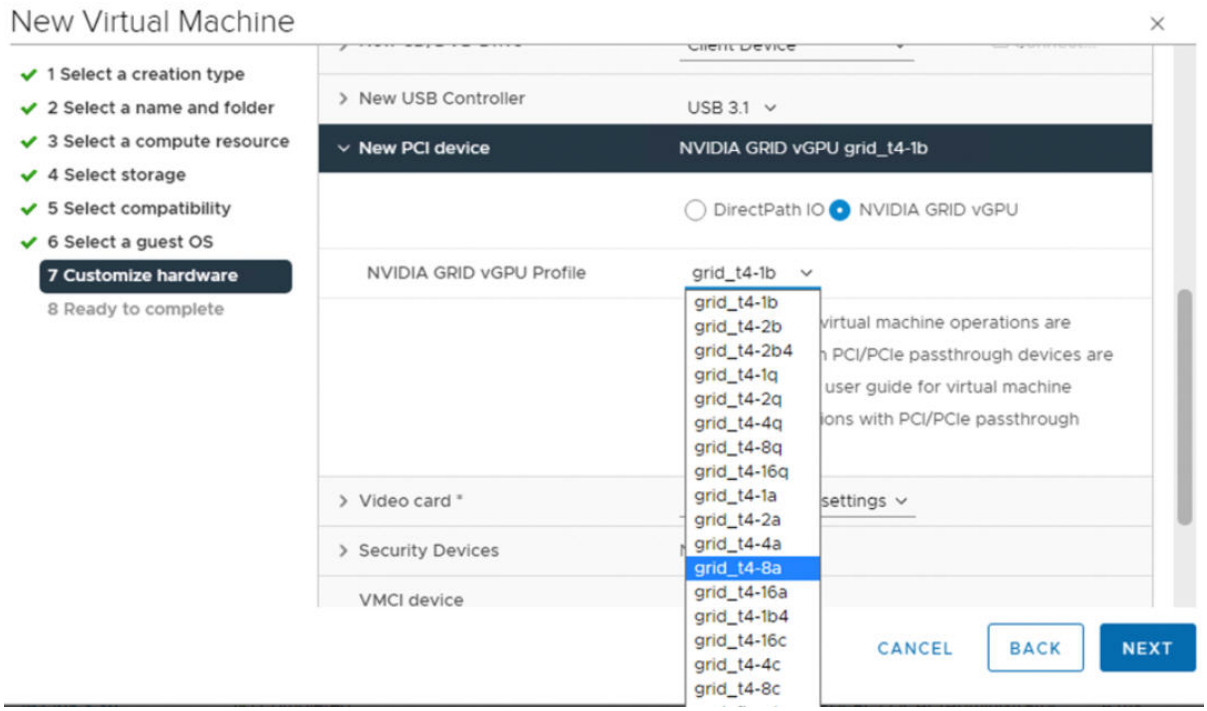
注： ESXi ホストにインストールされている NVAIE VIB のバージョンは、ノードの仮想マシンにインストールされている vGPU ソフトウェアのバージョンと一致する必要があります。以下のバージョンは単なる例です。

```
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
```

管理者の手順 5:ESXi ホストで NVIDIA vGPU 操作の準備ができていることを確認する

各 ESXi ホストで NVIDIA vGPU 操作を行う準備ができていることを確認するには、[ワークロード管理] を有効にする vCenter Server クラスタ内の各 ESXi ホストで次のチェックを実行します

- ESXi ホストに SSH 接続を行い、シェル モードに切り替えて、コマンド `nvidia-smi` を実行します。NVIDIA システム管理インターフェイスは、NVIDA vGPU ホスト マネージャから提供されるコマンド ライン ユーティリティです。このコマンドを実行すると、ホスト上の GPU とドライバが返されます。
- 次のコマンドを実行して、NVIDIA ドライバが適切にインストールされていることを確認します。`esxcli software vib list | grep NVIDA`
- ホストに GPU の直接共有が構成され、SR-IOV がオンになっていることを確認します（NVIDIA A30 デバイスまたは A100 デバイスを使用している場合）。
- vSphere Client を使用して、GPU 用に構成されている ESXi ホストに PCI デバイスを含む新しい仮想マシンを作成します。NVIDIA vGPU プロファイルが表示されて、選択可能になります。



管理者の手順 6：ワークロード管理を有効にする

[ワークロード管理] を有効にするには、TKG サービス クラスタのデプロイを参照してください。

注： [ワークロード管理] が有効になっている vSphere クラスタがすでに存在する場合は、クラスタが vGPU 用に構成された ESXi ホストを使用していると想定して、この手順をスキップします。

管理者の手順 7：TKR Ubuntu を含むコンテンツ ライブラリを作成または更新する

NVIDIA vGPU には Ubuntu オペレーティング システムが必要です。vGPU クラスタに Tanzu Kubernetes リリースの PhotonOS エディションを使用することはできません。

VMware では、Tanzu Kubernetes リリースの Ubuntu エディションが提供されています。vSphere 8 以降では、Ubuntu エディションはクラスタ YAML のアノテーションを使用して指定されます。

サポートされている Ubuntu TKR を使用して、既存のコンテンツ ライブラリを作成または更新します。5 章 TKG サービス クラスタ用 Kubernetes リリースの管理を参照してください。

注： vCenter Server で既存の TKR コンテンツ ライブラリがすでに構成されている場合は、この手順をスキップします。TKR 用に 2 つ目のコンテンツ ライブラリを作成しないでください。作成すると、システムが不安定になる可能性があります。

管理者の手順 8：vGPU プロファイルを使用するカスタム仮想マシン クラスを作成する

vGPU プロファイルを使用してカスタム仮想マシン クラスを作成します。次に、クラスタ仕様でこの仮想マシン クラスを使用して、TKGS クラスタ ノードを作成します。「NVIDIA vGPU デバイス用のカスタム仮想マシン クラスの作成」の手順を参照してください。

管理者の手順 9：vSphere 名前空間 を構成する

プロビジョニングする TKG vGPU クラスタごとに、vSphere 名前空間 を作成します。TKG サービス クラスタをホストするための vSphere 名前空間 の作成を参照してください。

編集権限を持つ vSphere SSO ユーザー/グループを追加して名前空間を構成し、パーシステント ボリュームにストレージ ポリシーを適用して、vSphere 名前空間 を構成します。TKG サービス クラスタ向けの vSphere 名前空間の構成を参照してください。

目的の Ubuntu イメージが格納されている TKR コンテンツ ライブラリを vSphere 名前空間 に関連付けます。TKR コンテンツ ライブラリの TKG サービス への関連付けを参照してください。

カスタム仮想マシン クラスを vSphere 名前空間 に関連付けます。

- vSphere 名前空間 の選択で、[仮想マシン サービス]タイルを選択し、[仮想マシン クラスの管理] をクリックします。
- クラスのリストで、作成したカスタム仮想マシン クラスを見つけます。
- クラスを選択し、[追加] をクリックします。

その他のガイダンスについては、仮想マシン クラスの vSphere 名前空間 への関連付けを参照してください。

管理者の手順 10：スーパーバイザー の準備ができていることを確認する

最後の管理タスクでは、スーパーバイザー がプロビジョニングされ、クラスタ オペレータが AI/ML ワークロード用の TKG クラスタをプロビジョニングする際に使用できることを確認します。

vCenter SSO 認証を使用した TKG サービス クラスタへの接続を参照してください。

TKG サービス クラスタへの AI/ML ワークロードのデプロイに関するクラスタ オペレータのワークフロー

開発者が TKG サービス クラスタに AI/ML ワークロードをデプロイできるようにするには、クラスタ オペレータとして、1 つ以上の Kubernetes クラスタを作成し、それぞれのクラスタに NVIDIA Network Operator と GPU Operator をインストールします。

Operator の手順 1：前提条件を確認する

これらの手順では、vSphere 管理者が NVIDIA GPU の環境を設定していることを前提としています。TKGS クラスタへの AI/ML ワークロードのデプロイに関する vSphere 管理者ワークフローを参照してください。

これらの手順では、GPU Operator の NVIDIA AI Enterprise (NVAIE) エディションをインストールしていることを前提としています。このエディションは、vSphere IaaS control plane で使用するため事前設定および最適化されています。NVAIE GPU Operator は、パブリック NGC カタログで使用可能な GPU Operator とは異なります。詳細については、NVIDIA AI Enterprise を参照してください。

これらの手順では、ESXi 用の VIB が一致する NVAIE GPU Operator と vGPU ドライバのバージョンを使用していることを前提としています。詳細については、NVIDIA GPU Operator Versioning を参照してください。

TKG クラスタをプロビジョニングする場合、TKR の Ubuntu エディションを使用する必要があります。vSphere 8 スーパーバイザー 上の TKG では、Ubuntu エディションはアノテーションを使用してクラスタ YAML で指定されます。

Operator の手順 2：NVIDIA vGPU 用 TKGS クラスタをプロビジョニングする

VMware は、NVIDIA GPU Operator と NVIDIA Network Operator を使用する NVIDIA GPU 認定サーバにおける NVIDIA 仮想 GPU のネイティブ TKGS サポートを提供します。これらのオペレータは、TKGS ワークロード クラスタにインストールします。vGPU ワークロードをホストするために TKGS クラスタをプロビジョニングするには、次の手順を実行します。

1 vSphere 向け Kubernetes CLI Tools をインストールします。

vSphere 向け Kubernetes CLI Tools のインストールを参照してください。

2 kubectl 向けの vSphere プラグイン を使用して、スーパーバイザー での認証を行います。

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

注： FQDN は、スーパーバイザー が有効になっている場合にのみ使用できます。

3 kubectl を使用して、vSphere 管理者が TKGS vGPU クラスタ用に作成した vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config get-contexts
```

```
kubectl config use-context TKG-GPU-CLUSTER-NAMESPACE
```

4 vSphere 管理者が作成した vGPU プロファイルを使用するカスタム仮想マシン クラスの名前を取得します。

```
kubectl get virtualmachineclass
```

注： 仮想マシン クラスをターゲット vSphere 名前空間 にバインドする必要があります。

5 vSphere 管理者がコンテンツ ライブラリから同期し、vSphere 名前空間 に追加した Ubuntu Tanzu Kubernetes リリース の TKR NAME を取得します。

```
kubectl get tkr
```

6 vGPU 対応 TKG クラスタをプロビジョニングするための YAML を作成します。

a 使用する TKGS クラスタのプロビジョニング API（v1alpha3 API または v1beta1 API）を決定します（「TKG クラスタのプロビジョニング API」を参照）。

b 選択した API に応じて、その API に関する Ubuntu の例を参照してください。

- v1alpha3 の例：TKC と Ubuntu TKR を使用する TKC
- v1beta1 の例：Ubuntu TKR を使用するクラスタ

注： Ubuntu OS イメージを使用する必要があります。Photon OS は使用できません。

c 前述のコマンドの出力から収集した情報を使用して、TKGS クラスタの仕様をカスタマイズします。

7 次の kubectl コマンドを実行して、クラスタをプロビジョニングします。

```
kubectl apply -f CLUSTER-NAME.yaml
```

例：

```
kubectl apply -f tkg-gpu-cluster-1.yaml
```

8 クラスタのプロビジョニングを確認します。

kubectl を使用して、クラスタ ノードのデプロイを監視します。

```
kubectl get tanzukubernetesclusters -n NAMESPACE
```

9 kubectl 向けの vSphere プラグイン を使用して、TKGS vGPU クラスタにログインします。

```
kubectl vsphere login --server=IP-ADDRESS-or-FQDN --vsphere-username USERNAME \
--tanzu-kubernetes-cluster-name CLUSTER-NAME --tanzu-kubernetes-cluster-namespace
NAMESPACE-NAME
```

10 クラスタを確認します。

次のコマンドを使用して、クラスタを確認します。

```
kubectl cluster-info
```

```
kubectl get nodes
```

```
kubectl get namespaces
```

```
kubectl api-resources
```

Operator の手順 3：NVIDIA Network Operator をインストールする

NVIDIA Network Operator では、Kubernetes カスタム リソースと Operator フレームワークを活用して、vGPU のネットワークを最適化します。詳細については、「NVIDIA Network Operator」を参照してください。

1 TKGS vGPU ワークロード クラスタにログイン済みであり、コンテキストが TKGS vGPU ワークロード クラスタ名前空間に設定されていることを確認します。

必要に応じて、「Operator の手順 2：NVIDIA vGPU 用 TKGS クラスタをプロビジョニングする」の手順を参照してください。

2 Helm ドキュメントを参照して、Helm をインストールします。

3 NVIDIA Network Operator Helm Chart を取得します。

```
helm fetch https://helm.ngc.nvidia.com/nvaie/charts/network-operator-v1.1.0.tgz --
username='$oauthtoken' --password=<YOUR API KEY> --untar
```

4 構成値の YAML ファイルを作成します。

```
vi values.yaml
```

5 `values.yaml` ファイルに次の情報をポピュレートします。

```
deployCR: true
 ofedDriver:
  deploy: true
 rdmaSharedDevicePlugin:
  deploy: true
  resources:
   - name: rdma_shared_device_a
   vendors: [15b3]
   devices: [ens192]
```

6 次のコマンドを使用して NVIDIA Network Operator をインストールします。

```
helm install network-operator -f ./values.yaml -n network-operator --create-namespace --
wait network-operator/
```

Operator の手順 4：NVIDIA GPU Operator をインストールする

NVIDIA では、NVIDIA AI Enterprise ユーザー用に事前構成された GPU Operator が提供されています。これらの手順は、この事前構成されたバージョンの GPU Operator を使用していることを前提としています。これらの手順は、NVIDIA によって提供されている GPU Operator のインストールの手順に基づいていますが、vSphere 8 の TKG 向けに更新されています。

次の手順を実行して、プロビジョニングした TKG クラスタに GPU Operator NVIDIA AI Enterprise をインストールします。

1 TKGS vGPU ワークロード クラスタにログイン済みであり、コンテキストが TKGS vGPU ワークロード クラスタ名前空間に設定されていることを確認します。

必要に応じて、「Operator の手順 2：NVIDIA vGPU 用 TKGS クラスタをプロビジョニングする」の手順を参照してください。

2 Helm ドキュメントを参照して Helm をインストールします（まだインストールされていない場合）。

3 `gpu-operator` Kubernetes 名前空間を作成します。

```
kubectl create namespace gpu-operator
```

4 空の vGPU ライセンス構成ファイルを作成します。

```
sudo touch gridd.conf
```

5 NLS クライアント ライセンス トークンを生成してダウンロードします。

『NVIDIA License System User Guide』の Section 4.6. Generating a Client Configuration Token を参照してください。

6 ダウンロードした NLS クライアント ライセンス トークンの名前を `client_configuration_token.tok` に変更します。

7 `gpu-operator` 名前空間に `licensing-config` ConfigMap オブジェクトを作成します。

この ConfigMap に、vGPU ライセンス構成ファイル (`gridd.conf`) と NLS クライアント ライセンス トークン (`*.tok`) を含めます。

```
kubectl create configmap licensing-config \
    -n gpu-operator --from-file=gridd.conf --from-file=<path>/
client_configuration_token.tok
```

8 NVIDIA GPU Operator で使用する Linux 用のコンテナ化された NVIDIA vGPU ソフトウェア グラフィック ドライバを含むプライベート レジストリ用のイメージ プル シークレットを作成します。

gpu-operator 名前空間で、レジストリ シークレット名 ngc-secret とプライベート レジストリ名 nvcr.io/nvaie のイメージ プル シークレットを作成します。表示されるフィールドに NGC API キーとメール アドレスを含めます。

```
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 NVAIE GPU Operator バージョン 2.2 の Helm チャートをダウンロードします。

API キーを置き換えます。

```
helm fetchhttps://helm.ngc.nvidia.com/nvaie/charts/gpu-operator-2-2-v1.11.1.tgz--username=
'$oauthtoken' \
--password=<YOUR API KEY>
```

10 NVAIE GPU Operator バージョン 2.2 を TKG クラスタにインストールします。

```
helm install gpu-operator ./gpu-operator-2-2-v1.11.1.tgz -n gpu-operator
```

Operator の手順 5：AI/ML ワークロードをデプロイする

NVIDIA GPU Cloud カタログには、vGPU 対応 Tanzu Kubernetes クラスタで AI/ML ワークロードを実行する場合に使用できる既製のコンテナ イメージがいくつかあります。使用可能なイメージの詳細については、NGC のドキュメントを参照してください。

NVIDIA vGPU デバイス用のカスタム仮想マシン クラスの作成

NVIDIA GRID vGPU デバイス用のカスタム仮想マシン クラスを作成するには、このトピックを参照してください。

vGPU プロファイルを使用したカスタム仮想マシン クラスの作成（v8 U2 P03 以降）

NVIDIA vGPU（仮想グラフィック処理ユニット）を使用すると、複数の仮想マシン (VM) で単一の物理 GPU を共有できます。TKGS クラスタで vGPU を使用するには、カスタム仮想マシン クラスを定義します。このリリース以降、カスタム仮想マシン クラスを定義するための新しいウィザードが追加されました。カスタム仮想マシン クラスを定義するための vGPU プロファイルを使用したカスタム仮想マシン クラスの作成（v8 U2 以前）とは異なり、vGPU プロファイルは仮想マシン クラスで構成されるのではなく、デバイスから読み取られます。

仮想マシン オペレータは vCenter Server インベントリをポーリングして、スーパーバイザー がデプロイされている vSphere クラスタを構成する ESXi ホストにインストールされているすべての vGPU デバイスを取得します。vGPU デバイスは、そのプロファイルを定義します。vGPU デバイス名は、プロファイルがマルチインスタンス GPU (MIG) であるか、時刻の共有 GPU であるかを示します。MIG はコンピューティングをスライスし、単一の GPU で複数のワークロードを並行して実行できるようにします。時刻の共有は GPU への共有アクセスを提供します。MIG モードは新しい GPU アーキテクチャに基づいていて、NVIDIA A100 デバイスおよび A30 デバイスでのみサポートされます。詳細については、NVIDIA のドキュメントを参照してください。

たとえば、GPU デバイス「grid-a100-40c」は、メモリが 40 GB の NVIDIA A100 GPU デバイスを仮想マシンに割り当てる時刻の共有 vGPU プロファイルを提供します。これに相当する MIG ベースの vGPU プロファイルは「grid-a100-7-40c」デバイスです。デバイスと RAM の間に数字が追加されているため、これが MIG プロファイルであることがわかります。「7」は、GPU デバイスに 7 つのコンピューティング スライスがあることを示します。MIG ベースの vGPU プロファイルには、1、2、3、または 7 個のコンピューティング スライスを含めることができます。

1 vSphere Client ホーム メニューから、[ワークロード管理] - [サービス] の順に選択します。

2 [仮想マシン クラス] タブを選択します。

3 [仮想マシン クラスの作成] をクリックします。

このアクションにより、仮想マシン クラスの作成手順について説明する [仮想マシン クラスの作成] ウィザードが起動します。

4 [名前] に仮想マシン クラスの名前を入力して、[次へ] をクリックします。

仮想マシン クラス名は仮想マシン クラスを識別します。次の要件を満たす一意の DNS 準拠名を入力します。

- 環境内のデフォルトまたはカスタム仮想マシン クラスの名前と重複しない、一意の名前を使用します。
- 63 文字以下の英数字の文字列を使用します。
- 大文字や空白は使用しないでください。
- ダッシュは最初または最後の文字以外の場所で使用します。たとえば、**`vm-class1`** のようにします。
- 仮想マシン クラスを作成した後に、名前を変更することはできません。

5 [互換性] で [ESXi 8.0 U2 以降] を選択して、[次へ] をクリックします。

詳細については、「仮想マシンの互換性」を参照してください。

注： 仮想マシン クラスの作成後に、仮想マシン クラスのハードウェア互換性を変更することはできません。

6 [構成] - [仮想ハードウェア] で、NVIDIA GPU デバイスを仮想マシン クラスに追加します。

a [構成] - [仮想ハードウェア] - [新規デバイスを追加] - [PCI デバイス] の順に選択します。

b リストから目的の NVIDIA GRID vGPU デバイスを選択します。NVIDIA GRID vGPU プロファイルには、[時刻の共有] と [マルチインスタンス GPU 共有] の 2 種類があります。デバイスを選択すると、プロファイルがシステムによって検出されます。

注： MIG タイプのプロファイルの NVIDIA GRID vGPU デバイスは 1 つのみ仮想マシン クラスに追加できます。

c [選択] をクリックします。[仮想ハードウェア] タブに [新規 PCI デバイス] が表示されます。

7 [構成] - [仮想ハードウェア] で、[CPU]、[メモリ]、[新規 PCI デバイス]、[ビデオ カード]、および [セキュリティ デバイス] に目的の設定を指定します。

表 13-3. CPU の構成

設定	構成
CPU	仮想マシンの仮想 CPU 数を選択します。詳細については、「仮想 CPU の構成と制限」を参照してください。
CPU トポロジ	パワーオン時に自動割り当て
予約	予約は 0 ～ 10 MHz にする必要があります
制限	制限は 10 MHz 以上にする必要があります
シェア	オプションは、低、標準、高、カスタムです
ハードウェア仮想化	ハードウェア アシストによる仮想化をゲスト OS に公開するには、このオプションを選択します
パフォーマンス カウンタ	仮想 CPU パフォーマンス カウンタの有効化
スケジュール設定のアフィニティ	この仮想マシンに対する物理プロセッサのアフィニティを選択します。範囲を指定する場合は「-」を使用します。また、値を区切る場合は「,」を使用します。たとえば、「0, 2, 4-7」と入力すると、プロセッサ 0、2、4、5、6、7 が指定されます。文字列をクリアしてアフィニティ設定を削除します。
I/O MMU	メモリ管理ユニット（ページからディスク）を有効にする場合に選択します

表 13-4. メモリの構成

設定	構成
メモリ	仮想マシンのメモリのサイズを選択します。詳細については、「仮想マシンの最大メモリ」を参照してください。
予約	仮想マシンに保証される最小割り当てを指定するか、すべてのゲスト メモリを予約します。予約を満たすことができない場合、仮想マシンは実行できません。
制限	制限するメモリの量を選択して、仮想マシンのメモリ使用量に制限を設定します。
シェア	共有するメモリの量を選択します。シェアは、割り当てるメモリ容量の相対的なメトリックを表します。詳細については、「メモリの共有」を参照してください。
メモリ ホット プラグ	パワーオン状態の仮想マシンにメモリ リソースを追加できるようにする場合は有効（オン）にします。詳細については、「メモリのホット アド設定」を参照してください。

表 13-5. [新規 PCI デバイス] > [GPU 共有] の構成

時刻の共有モード	MIG モード
時刻の共有モードの場合、vGPU スケジューラは vGPU 間でパフォーマンスを調整するというベスト エフォート型の目標に従い、vGPU 対応の各仮想マシンの作業を一定期間、順番に実行するように GPU に指示します。	MIG モードの場合は、単一の GPU デバイスで複数の vGPU 対応仮想マシンを並列に実行できます。MIG オプションが表示されない場合、選択した PCI デバイスではサポートされていません。

表 13-6. ビデオ カードの構成

設定	構成
ビデオ カード	ハードウェアから設定を自動検出するか、カスタム設定を入力するかを選択します。自動検出を選択すると、他の設定は構成できません。
ディスプレイ数	ディスプレイの数を選択します。
ビデオ メモリの合計	ビデオ メモリの合計を MB 単位で入力します。
3D グラフィックス	3D サポートを有効にする場合に選択します。

表 13-7. セキュリティ デバイスの構成

設定	構成
セキュリティ デバイス	SGX セキュリティ デバイスがインストールされている場合は、ここで仮想マシン設定を構成できます。それ以外の場合は、このフィールドを構成できません。詳細については、SGX のドキュメントを参照してください。

8 [構成] - [仮想マシン オプション] タブを選択し、追加の仮想マシン設定を構成します。ガイダンスについては、「仮想マシン オプションの設定」を参照してください。

9 [構成] - [詳細パラメータ] タブを選択し、仮想マシン クラスの属性を追加します。

10 [次へ] をクリックします。

11 [確認] 画面で詳細を確認して、[完了] をクリックします。

12 新しい仮想マシン クラスを vSphere 名前空間に関連付けます。仮想マシン クラスの vSphere 名前空間 への関連付けを参照してください。

図 13-1. NVIDIA vGPU デバイスの選択

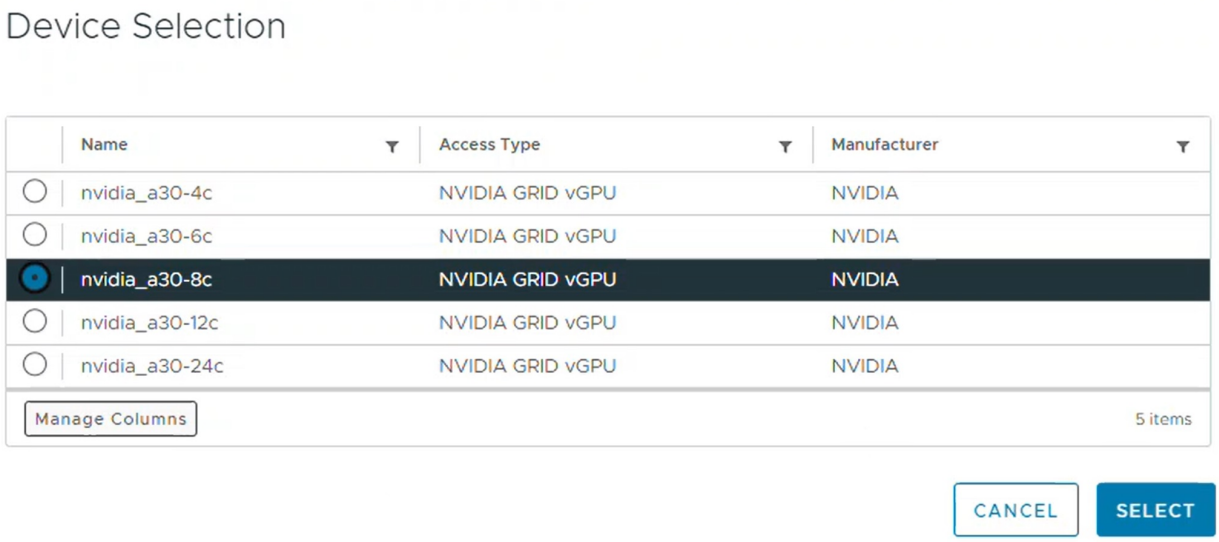
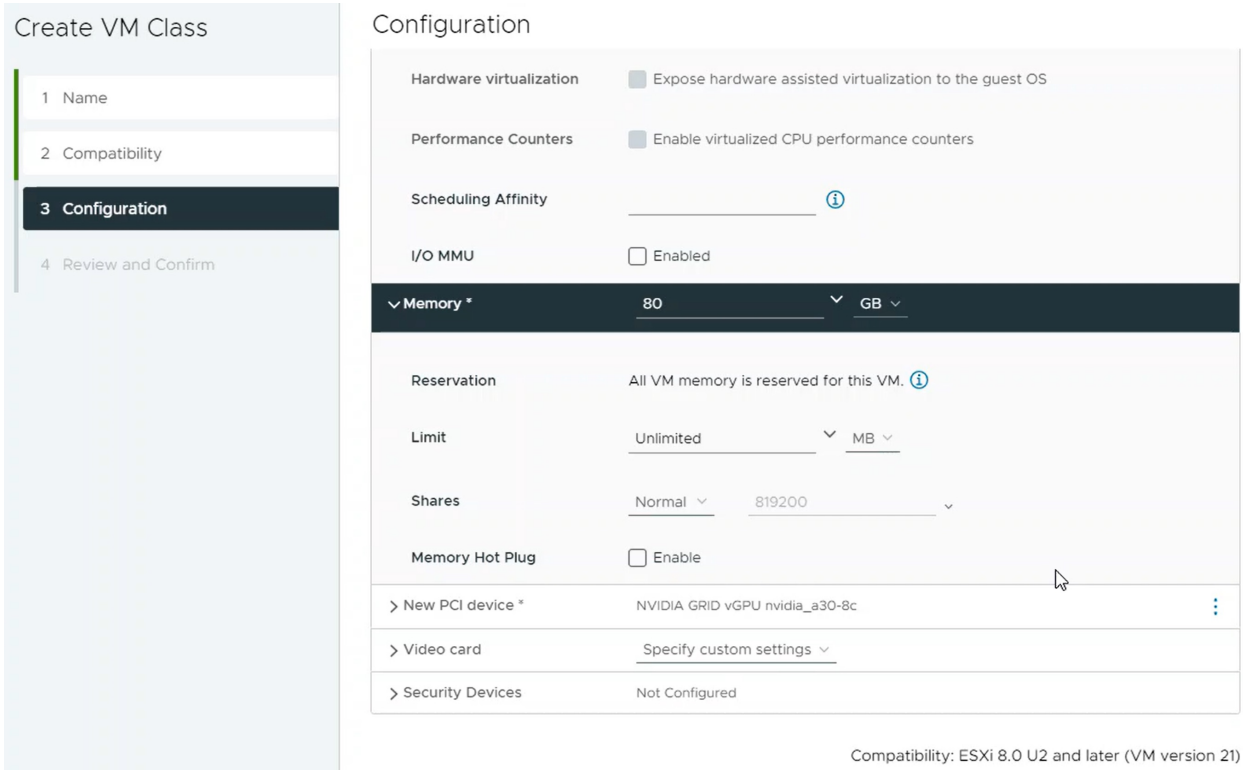


図 13-2. NVIDIA vGPU の [新規 PCI デバイス]



vGPU プロファイルを使用したカスタム仮想マシン クラスの作成 (v8 U2 以前)

次の手順では、vGPU プロファイルを使用するカスタム仮想マシン クラスを作成します。TKG クラスタ ノードが作成されるときに、このクラス定義が使用されます。

次の手順に従って、vGPU プロファイルを使用するカスタム仮想マシン クラスを作成します。

1 vSphere Client を使用して、vCenter Server にログインします。

2 [ワークロード管理] を選択します。

3 [[サービス]] を選択します。

4 [仮想マシン クラス] を選択します。

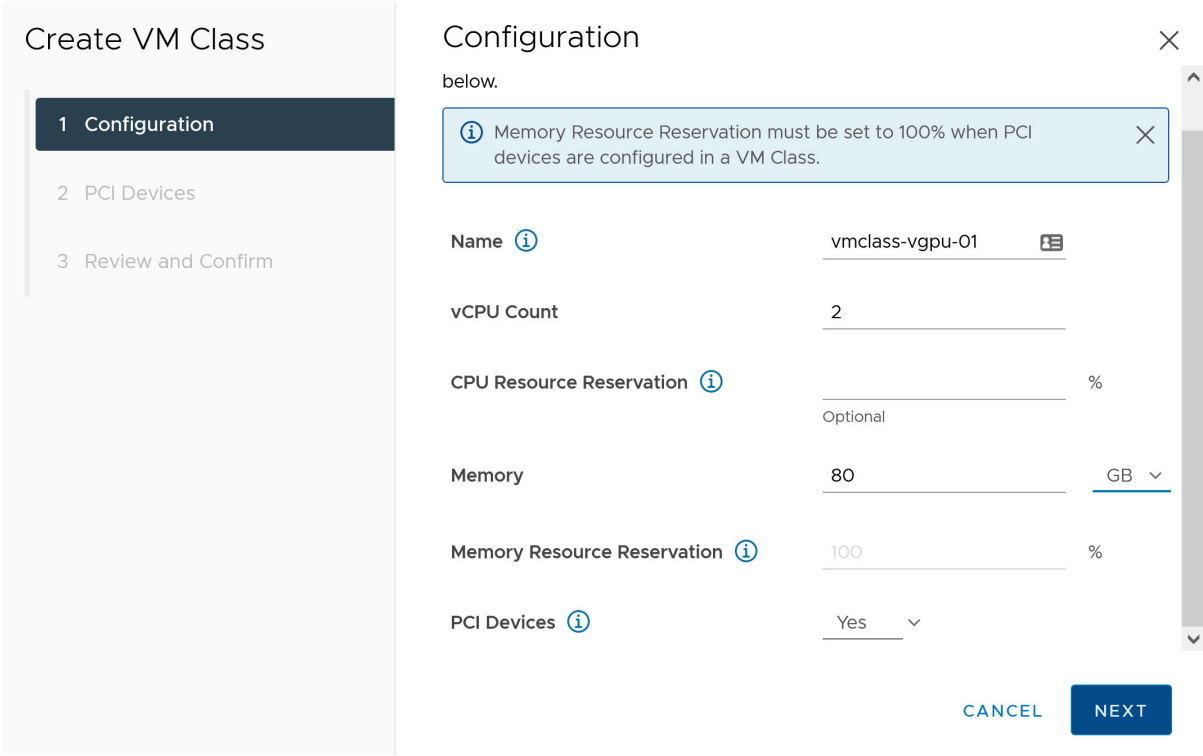
5 [仮想マシン クラスの作成] をクリックします。

6 [構成] タブで、カスタム仮想マシン クラスを構成します。

構成フィールド	説明
[名前]	カスタム仮想マシン クラスのわかりやすい名前を入力します (**vmclass-vgpu-1** など)。
[vCPU の数]	**2**
[CPU リソース予約]	オプション。空白のままにする場合は [OK] をクリックします。
[メモリ]	**80** [GB] など

構成フィールド	説明
[メモリ リソース予約]	[100%]（仮想マシン クラスで PCI デバイスが構成されている場合は必須）
[PCI デバイス]	[はい] **注：** PCI デバイスに対して [はい] を選択すると、GPU デバイスを使用していることがシステムに通知され、vGPU 構成をサポートするように仮想マシン クラスの構成が変更されます。詳細については、「vSphere with Tanzu での仮想マシン クラスへの PCI デバイスの追加」を参照してください。

例：



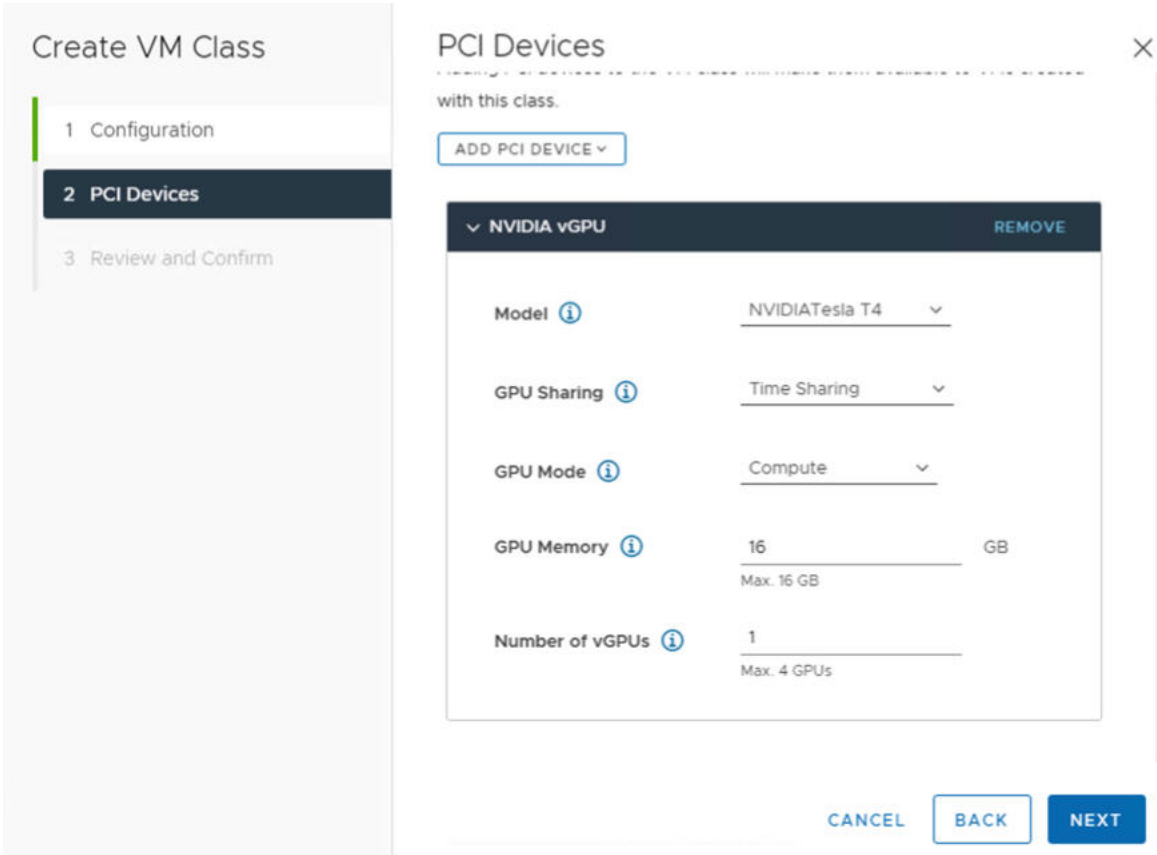
7 [次へ] をクリックします。

8 [PCI デバイス] タブで [PCI デバイスの追加] - [NVIDIA vGPU] オプションを選択します。

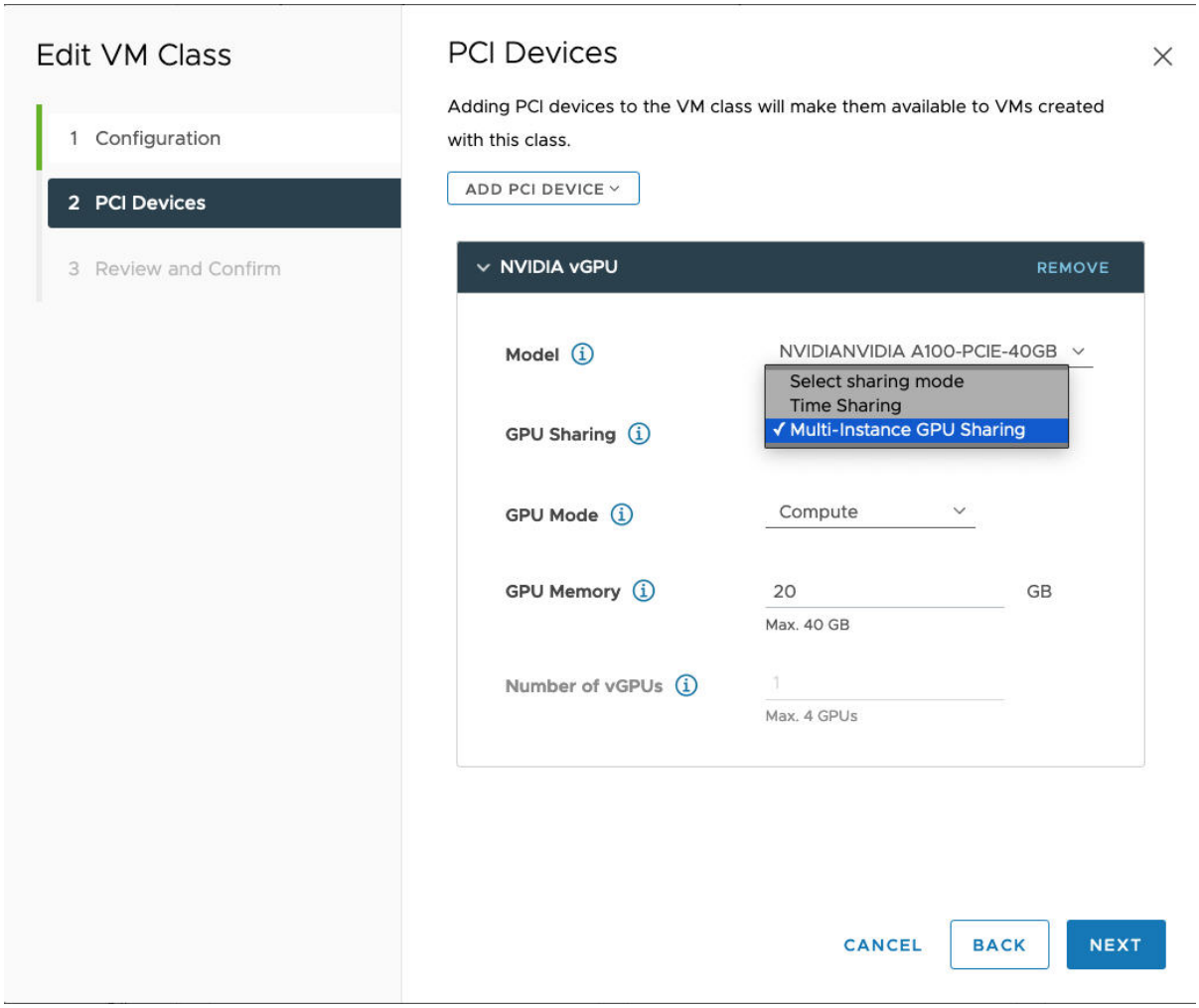
9 NVIDIA vGPU モデルを構成します。

NVIDIA vGPU フィールド	説明
[モデル]	[NVIDIA vGPU] - [モデル] メニューで使用可能なモデルの中から、NVIDIA GPU ハードウェア デバイス モデルを選択します。プロファイルが表示されない場合は、クラスタ内のどのホストも PCI デバイスをサポートしていません。
[GPU 共有]	この設定は、GPU 対応の仮想マシン間における GPU デバイスの共有方法を定義します。vGPU の実装には、[時刻の共有] と [マルチインスタンス GPU 共有]の 2 種類があります。 時刻の共有モードの場合、vGPU スケジューラは vGPU 間でパフォーマンスを調整するというベスト エフォート型の目標に従い、vGPU 対応の各仮想マシンの作業を一定期間、順番に実行するように GPU に指示します。 MIG モードの場合は、単一の GPU デバイスで複数の vGPU 対応仮想マシンを並列に実行できます。MIG モードは新しい GPU アーキテクチャに基づいていて、NVIDIA A100 デバイスおよび A30 デバイスでのみサポートされます。MIG オプションが表示されない場合、選択した PCI デバイスではサポートされていません。
[GPU モード]	[コンピューティング]
[GPU メモリ]	8 [GB] など
[vGPU の数]	[1] など

たとえば、時刻の共有モードで構成された NVIDIA vGPU プロファイルを次に示します。



たとえば、サポートされている GPU デバイスを使用して MIG モードで構成された NVIDIA vGPU プロファイルを次に示します。



10 [次へ] をクリックします。

11 選択内容を確認します。

12 [終了] をクリックします。

13 新しいカスタム仮想マシン クラスが仮想マシン クラスのリストで使用可能になっていることを確認します。

[vGPU と動的 DirectPath I/O]

vGPU と動的 DirectPath I/O を使用している場合、以下の追加構成を実行します。2 つ目の PCI デバイス構成を [動的 DirectPath I/O] を指定し、サポートされている PCI デバイスを選択して作成したカスタム仮想マシン クラスに追加します。このタイプの仮想マシン クラスがインスタンス化されている場合、仮想マシンの配置は vSphere Distributed Resource Scheduler (DRS) によって決まります。

1 [ワークロード管理] を選択します。

2 [[サービス]] を選択します。

3 [仮想マシン クラス] を選択します。

4 [NVIDIA vGPU] プロファイルを使用してすでに構成されているカスタム仮想マシン クラスを編集します。

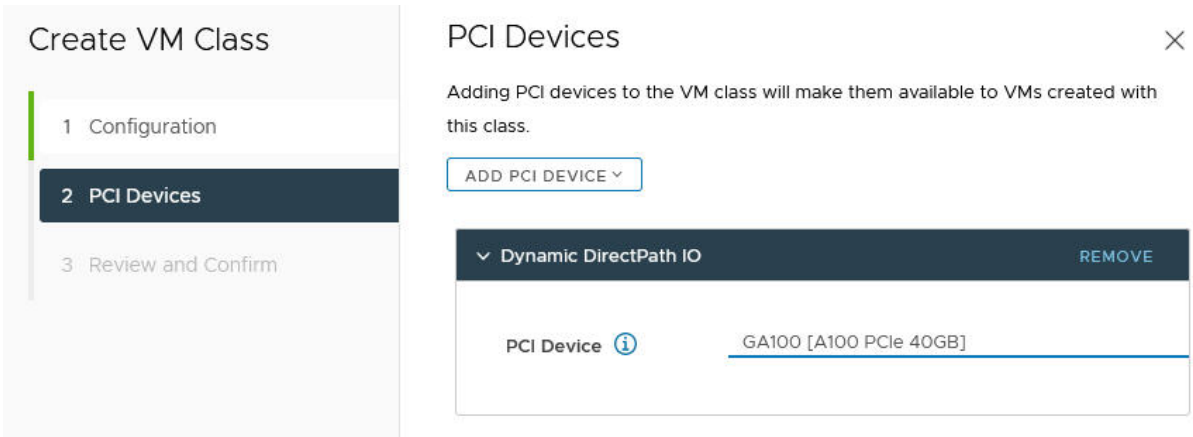
5 [PCI デバイス] タブを選択します。

6 [PCI デバイスの追加] をクリックします。

7 [動的 DirectPath I/O] オプションを選択します。

8 [PCI デバイス] を選択します。

例：



9 [次へ] をクリックします。

10 選択内容を確認します。

11 [終了] をクリックします。

12 新しいカスタム仮想マシン クラスが仮想マシン クラスのリストで使用可能になっていることを確認します。

TKG サービス クラスタでのプライベート レジストリの使用

14

コンテナ レジストリは、Kubernetes オペレータにコンテナ イメージの保存と共有に便利なリポジトリを提供します。このセクションでは、TKG サービス クラスタでプライベート レジストリを使用する方法について説明します。

次のトピックを参照してください。

- TKG サービス クラスタとプライベート コンテナ レジストリの統合
- プライベート レジストリ認証情報シークレットの作成
- プライベート レジストリでのコンテナ イメージからのポッドの作成
- Harbor スーパーバイザー サービスのインストール
- Harbor レジストリ証明書を使用した Docker クライアントの構成
- プライベート Harbor レジストリへの標準パッケージのプッシュ

TKG サービス クラスタとプライベート コンテナ レジストリの統合

TKG サービス クラスタをプライベート コンテナ レジストリと統合するには、このトピックを参照してください。

プライベート コンテナ レジストリの使用事例

コンテナ レジストリは、Kubernetes 環境に重要な機能を提供し、コンテナ イメージを保存して共有するための一元的なリポジトリとして機能します。最もよく使用されるパブリック コンテナ レジストリは Docker Hub です。プライベート コンテナ レジストリのサービスは多数あります。VMware Harbor は、スーパーバイザー に付属するオープンソースのクラウド ネイティブなプライベート コンテナ レジストリです。

プライベート コンテナ レジストリの統合

プライベート レジストリを TKG サービス クラスタと統合するには、1 つ以上の自己署名 CA 証明書を使用して、HTTPS 経由でプライベート レジストリ コンテンツを提供するようにクラスタを構成します。そのためには、クラスタ仕様の `trust` フィールドに `additionalTrustedCAs` 値を設定します。TKGS クラスタが信頼する自己署名 CA 証明書は、いくつでも定義できます。この機能を使用すると、証明書のリストを簡単に定義し、ローテーションが必要な証明書を更新できます。

最初にクラスタを作成するときにプライベート レジストリ証明書を構成することも、既存のクラスタを更新してプライベート レジストリ証明書を指定することもできます。既存のクラスタを編集し、プライベート レジストリ証明書フィールドを追加するには、`kubectl edit` メソッドを使用してください。kubectl のテキスト エディタの構成を参照してください。

`trust.additionalTrustedCAs` フィールドの実装は、TKGS クラスタのプロビジョニング用にサポートされている API 間で若干異なることに注意してください。詳細については、「表 14-1. v1alpha3 API の trust フィールド」および「表 14-2. v1beta1 API の trust 変数」を参照してください。

v1alpha3 API の例

次の例は、v1alpha3 API を使用してプロビジョニングされた TKG サービス クラスタを、CA 証明書を使用してプライベート コンテナ レジストリと統合する方法を示しています。

TanzuKubernetesCluster v1alpha3 API では、`trust.additionalTrustedCAs` フィールドには 1 つ以上の名前とデータのペアが含まれており、それぞれにプライベート レジストリの TLS 証明書を含めることができます。

表 14-1. v1alpha3 API の trust フィールド

フィールド	説明
`trust`	セクション マーカー。データを受け入れない。
`additionalTrustedCAs`	セクション マーカー。それぞれの `name` と `data` を示す証明書の配列が含まれます。
`name`	CA 証明書のユーザー定義名。
`data`	double 型の base64 エンコードの PEM 形式の CA 証明書 (`ca.crt`) の内容。 **注：** v1alpha3 API では、証明書の内容を single 型の base64 エンコード形式にする必要があります。内容が single 型の base64 エンコード形式でない場合は、結果の PEM ファイルを処理できません。

Harbor レジストリ証明書を使用して Harbor を v1alpha3 API クラスタと統合するには、次の手順を実行します。

1 Harbor Web インターフェイスの [プロジェクト] - [リポジトリ] 画面で Harbor [レジストリ証明書] をダウンロードします。

   CA 証明書ファイルが `ca.crt` としてダウンロードされます。

2 CA 証明書の内容を single 型の base64 エンコード形式にします。

   - Linux：`base64 -w 0 ca.crt`
   - Windows：https://www.base64encode.org/

3 クラスタ仕様に `trust.additionalTrustedCAs` フィールドを含めて、`name` と `data` の値をポピュレートします。

```
#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 証明書をローテーションするには、クラスタ仕様に対して `kubectl edit` を実行し、`trust.additionalTrustedCAs.data` 値を更新してからローリング アップデートを開始します。

v1beta1 API の例

次の例では、v1beta1 API を使用してプロビジョニングされた TKG サービス クラスタを、CA 証明書を使用してプライベート コンテナ レジストリと統合する方法について説明します。

プライベート コンテナ レジストリを、クラスタ v1beta1 API でプロビジョニングされた TKGS クラスタと統合するには、trust 変数を使用して、プライベート レジストリ証明書を含む Kubernetes シークレットをポピュレートします。

表 14-2. v1beta1 API の trust 変数

フィールド	説明
`trust`	セクション マーカー。データを受け入れない。
`additionalTrustedCAs`	セクション マーカー。それぞれの名前を示す証明書の配列が含まれます。
`name`	double 型の base64 エンコードの PEM 形式の CA 証明書を含む Kubernetes シークレットの `data` マップ フィールドのユーザー定義名。 **注：** v1beta1 API では、証明書の内容を double 型の base64 エンコード形式にする必要があります。内容が double 型の base64 エンコード形式でない場合は、結果の PEM ファイルを処理できません。

Harbor レジストリ証明書を使用して Harbor を v1beta1 API クラスタと統合するには、次の手順を実行します。

1 Harbor Web インターフェイスの [プロジェクト] - [リポジトリ] 画面で Harbor [レジストリ証明書] をダウンロードします。

CA 証明書ファイルが `ca.crt` としてダウンロードされます。

2 CA 証明書の内容を double 型の base64 エンコード形式にします。

- Linux：`base64 -w 0 ca.crt | base64 -w 0`
- Windows：https://www.base64encode.org/

3 次の内容の Kubernetes シークレット定義 YAML ファイルを作成します。

```
#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
```

ここで、

- シークレットの `data` マップの値は、ユーザー定義の文字列です。これは CA 証明書の名前（この例では `additional-ca-1`）であり、値は double 型の base64 エンコード形式の証明書です。
- `metadata` セクションで、シークレット *`CLUSTER-NAME`*`-user-trusted-ca-secret` に名前を付けます。*CLUSTER-NAME* はクラスタの名前です。このシークレットは、クラスタと同じ vSphere 名前空間 で作成する必要があります。

4 Kubernetes シークレットを宣言によって作成します。

```
kubectl apply -f additional-ca-1.yaml
```

5 シークレットの作成を確認します。

```
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 シークレットのデータ マップの名前を参照するクラスタ仕様に `trust` 変数を含めます。この例では、`additional-ca-1` です。

```
#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 証明書をローテーションするには、新しいシークレットを作成し、適切な値を使用してクラスタ仕様を編集します。これにより、クラスタのローリング アップデートがトリガされます。

注： システムは、*CLUSTER-NAME*`-user-trusted-ca-secret` への変更を監視しません。`data` マップ値が変更された場合は、クラスタに反映されません。新しいシークレットを作成して、そのデータ マップの `name` を `trust.additionalTrustCAs` にする必要があります。

プライベート レジストリ認証情報シークレットの作成

レジストリ認証情報シークレットを作成し、ポッドとデプロイ仕様からエラーなしでコンテナ イメージをプルできるように参照します。

Docker Hub には、イメージをプルする元になるアカウントが必要です。Docker Hub 認証情報を使用して Kubernetes シークレットを作成し、ポッドとデプロイ仕様からこのシークレットを参照できます。

この方法は、他のプライベート レジストリにも使用できます。

前提条件

Ubuntu クライアント コンピュータに Docker をインストールします。https://docs.docker.com/engine/install/ubuntu/を参照してください。

手順

1 `docker version` を実行し、Docker がインストールされていることを確認します。

2 `docker login -u USERNAME` を実行します。

3 Docker Hub のパスワードを入力します。

4 `cat ~/.docker/config.json` を実行します。

5 次のコマンドを実行して、Docker Hub アクセス認証情報を含む `regcred` という名前の汎用シークレットを作成します。

```
kubectl create secret generic regcred \
    --from-file=.dockerconfigjson=/home/ubuntu/.docker/config.json \
    --type=kubernetes.io/dockerconfigjson
```

`secret/regcred created` が表示されます。

6 シークレットを確認します。

```
kubectl get secrets

NAME                  TYPE                                  DATA   AGE
default-token-w7wqk   kubernetes.io/service-account-token   3      6h28m
regcred               kubernetes.io/dockerconfigjson        1      3h22m
```

YAML で regcred シークレットを参照します。

7 コンテナ イメージのポッドまたはデプロイ仕様の `imagePullSecrets` セクションで、`regcred` シークレットを参照します。

例：

```
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
```

プライベート レジストリでのコンテナ イメージからのポッドの作成

TKG サービス クラスタのためにプライベート コンテナ レジストリからイメージをプルするには、プライベート レジストリの詳細を使用してワークロード YAML を構成します。

この手順は、Harbor レジストリ などのプライベート コンテナ レジストリからイメージをプルする場合に使用できます。この例では、Harbor レジストリ に保存されているイメージを使用するポッド仕様を作成し、以前に構成したイメージ プル シークレットを利用します。

前提条件

レジストリ認証情報シークレットを作成します。プライベート レジストリ認証情報シークレットの作成を参照してください

手順

1 TKG クラスタをプロビジョニングします。

Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。

2 クラスタにログインします。

Kubectl を使用した vCenter Single Sign-On ユーザーとしての TKG サービス クラスタへの接続を参照してください。

3 レジストリ認証情報シークレットを作成します。

プライベート レジストリ認証情報シークレットの作成を参照してください。

4 プライベート レジストリの詳細を含むサンプルのポッド仕様を作成します。

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>
```

- <workload-name> をポッド ワークロードの名前に置き換えます。
- <kubernetes-namespace> を、ポッドが作成されるクラスタ内の Kubernetes 名前空間に置き換えます。この名前空間は、レジストリ サービスのイメージ プル シークレットが格納されている、Tanzu Kubernetes クラスタ内の Kubernetes 名前空間（デフォルトの名前空間など）と同じにする必要があります。
- <Registry-IP-or-FQDN> を、スーパーバイザー で実行されている Harbor レジストリ インスタンスの IP アドレスまたは FQDN に置き換えます。
- *<vsphere-namespace>* を、ターゲット TKG クラスタがプロビジョニングされている vSphere 名前空間 に置き換えます。
- <image-name> を、任意のイメージ名に置き換えます。
- <version> を、イメージの適切なバージョン（「最新」バージョンなど）に置き換えます。
- <registry-secret-name> を、以前に作成したレジストリ サービスのイメージ プル シークレットの名前に置き換えます。

5 定義したポッド仕様を使用してワークロードを作成します。

```
kubectl apply -f pod-example.yaml
```

ポッドは、レジストリからプルされたイメージに基づいて作成する必要があります。

Harbor スーパーバイザー サービスのインストール

Harbor コンテナ レジストリをスーパーバイザー サービスとしてインストールし、プライベート レジストリとして Harbor を実行できます。

前提条件

次の前提条件を満たす必要があります。

- vSphere 8 スーパーバイザー が有効である
- スーパーバイザー サービスについて理解している

注： ここで示す手順は、vSphere 8 および NSX 4 ネットワークで検証されています。

必要な YAML ファイルのダウンロード

Contour や Harbor などの必要な YAML ファイルをダウンロードします。

1 Kubernetes Ingress コントローラ サービスのサイトから次の Contour ファイルをダウンロードします。

a Contour サービス定義ファイル：contour.yml

b Contour サービス構成ファイル：`contour-data-values.yml`

2 Cloud Native Registry Service のサイトから次の Harbor ファイルをダウンロードします。

a Harbor サービス定義ファイル：`harbor.yml`

b Harbor サービス構成ファイル：`harbor-data-values.yml`

Contour のインストール

Harbor をインストールする前に、Contour をインストールする必要があります。

1 [ワークロード管理] - [] - [サービス] - [追加] で、vCenter Server に `contour.yml` をアップロードします。

2 Contour サービス定義が追加されたことを確認します。

3 [ワークロード管理] - [スーパーバイザー] - [スーパーバイザー] - [構成] の順に選択します。

4 [スーパーバイザー サービス] - [概要] の順に選択します。

5 [使用可能] タブを選択します。

6 [Contour] を選択し、[インストール] をクリックします。

7 `contour-data-values.yml` の内容をコピーして [YAML サービス構成] 入力フィールドに貼り付けます。

注： Contour データ値はそのまま使用できます。構成を変更する必要はありません。Envoy のサービス タイプは LoadBalancer に設定されています。

8 [OK] をクリックして、Contour のインストールを続行します。

9 Contour がインストールされたことを確認します。

a *svc-contour-domain-XXXX* という名前の vSphere 名前空間 を選択します。

b [ネットワーク] タブを選択し、[サービス] を選択します。

c ClusterIP タイプの contour サービスと LoadBalancer タイプの envoy サービスが表示されます。envoy サービスには外部 IP アドレスが必要です。

Harbor データ値の更新

Harbor をインストールする前に、データ値ファイルを更新します。

1 テキスト エディタを使用して `harbor-data-values.yml` ファイルを開きます。

2 以下の編集を行います（必要最小限の編集であり、他のフィールドの編集は任意）。

3 変更を保存します。

名前	値
hostname	harbordomain.com（一意のホスト名を選択）
tlsCertificate.tlsSecretLabels	{"managed-by": "vmware-vRegistry"}（この値を確認してそのまま使用）

名前	値
persistence.persistentVolumeClaim.registry.storageClass	"vwt-storage-policy"（スーパーバイザーの vSphere ストレージ ポリシーの名前を入力）
persistence.persistentVolumeClaim.jobservice.storageClass	"vwt-storage-policy"（スーパーバイザーの vSphere ストレージ ポリシーの名前を入力）
persistence.persistentVolumeClaim.database.storageClass	"vwt-storage-policy"（スーパーバイザーの vSphere ストレージ ポリシーの名前を入力）
persistence.persistentVolumeClaim.redis.storageClass	"vwt-storage-policy"（スーパーバイザーの vSphere ストレージ ポリシーの名前を入力）
persistence.persistentVolumeClaim.trivy.storageClass	"vwt-storage-policy"（スーパーバイザーの vSphere ストレージ ポリシーの名前を入力）

Harbor のインストール

次の手順を実行して、Harbor をインストールします。

1 [ワークロード管理] - [] - [サービス] - [追加] で、vCenter Server に `harbor.yml` をアップロードします。

2 Harbor サービス定義が追加されたことを確認します。

3 [ワークロード管理] - [スーパーバイザー] - [スーパーバイザー] - [構成] の順に選択します。

4 [スーパーバイザー サービス] - [概要] の順に選択します。

5 [使用可能] タブを選択します。

6 [Harbor] を選択し、[インストール] をクリックします。

7 編集した `harbor-data-values.yml` の内容をコピーして [YAML サービス構成] 入力フィールドに貼り付けます。

8 [OK] をクリックして、Harbor のインストールを続行します。

9 Harbor がインストールされたことを確認します。

   a *svc-harbor-domain-XXXX* という名前の vSphere 名前空間 を選択します。

   b [ネットワーク] タブを選択し、[サービス] を選択します。

   c Harbor 用に複数のコンテナがインストールされます。それぞれのサービスのタイプは ClusterIP です。

Harbor 用の DNS の構成

ドメイン名を登録し、Harbor の DNS レコードを設定する必要があります。

1 [ワークロード管理] - [名前空間] の順に選択します。

2 [Contour] 名前空間を削除します。

3 [ネットワーク] - [サービス] の順に選択します。

4 Envoy Ingress サービスの外部 IP アドレス（10.197.154.71 など）を記録します。

5 Harbor 構成で指定した Harbor ドメイン名 (FQDN) を登録します。

6 AWS Route 53 または同様のサービスを使用して DNS「A」レコードを作成します。

Harbor へのログイン

Harbor DNS が設定されたら、ログインします。

1 Harbor に登録したドメイン名に移動します。

2 admin および Harbor 構成で指定したパスワードを使用してドメインにログインします。

3 ログインしたら、パスワードをより安全なものに変更します。

Harbor レジストリを信頼するためのスーパーバイザーの構成（オプション）

TKG クラスタと Harbor の両方が同じ スーパーバイザー にデプロイされている場合、TKG クラスタは Harbor スーパーバイザー サービスを信頼するように自動的に構成されます。ただし、スーパーバイザー は vSphere ポッド の作成時に Harbor スーパーバイザー サービスを信頼するように自動的に構成されません。Harbor CA 証明書を使用して configmap を更新することによって、スーパーバイザー と Harbor サービス間の信頼を確立するには、次の手順を実行します。

1 Harbor で、[管理] - [構成] - [システム設定] の順に移動します。

2 レジストリ ルート証明書（`ca.crt` という名前のファイル）をダウンロードします。

3 KUBE_EDITOR 環境変数を構成します。

kubectl のテキスト エディタの構成を参照してください。

4 kubectl を使用して スーパーバイザー にログインします。

Kubectl を使用した vCenter Single Sign-On ユーザーとしての スーパーバイザー への接続を参照してください。

5 コンテキストを スーパーバイザー コンテキスト（IP アドレス）に切り替えます。

6 次のコマンドを使用して configmap/image-fetch-ca-bundle を編集します。

```
kubectl edit configmap image-fetcher-ca-bundle -n kube-system
```

7 Harbor ca.crt ファイルの内容をコピーして、既存の証明書の下にある configmap に追加します（これは スーパーバイザー 用であるため、変更しないでください）。ファイルに加えた編集内容を保存します。Kubectl から「configmap/image-fetcher-ca-bundle edited」と報告されます。

Harbor レジストリ証明書を使用した Docker クライアントの構成

Docker を使用してレジストリ内のコンテナ イメージを操作するには、Docker クライアントにレジストリ証明書を追加します。この証明書は、レジストリへのログイン時の Docker の認証に使用されます。

Harbor レジストリや Docker Hub などのコンテナ レジストリと通信するように Docker クライアントを構成します。この例は、Harbor スーパーバイザー サービス を使用していることを前提としています。

前提条件

このタスクでは、Docker デーモンがインストールされている Linux ホスト (Ubuntu) を使用していることを前提としています。Ubuntu ホストに Docker Engine（デーモン）をインストールするには、https://docs.docker.com/engine/install/ubuntu/ を参照してください。

Docker がインストールされ、Docker Hub からイメージをプルできることを確認するには、次のコマンドを実行します。

```
docker run hello-world
```

予期される結果：

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

注： これらの手順は、Ubuntu 20.04 および Docker 19.03 を使用して検証されています。

手順

1 Harbor レジストリにログインします。

2 [管理] - [構成] - [レジストリ ルート証明書] の順に選択します。

3 [ダウンロード] をクリックして、`ca.crt` という名前の Harbor レジストリ証明書をダウンロードします。

注： 必要に応じて、証明書の名前を `ca.crt` に変更します。

4 `ca.crt` ファイルを Docker ホスト クライアントに安全にコピーします。

5 Docker ホストで、Harbor の IP アドレスを使用してプライベート レジストリのディレクトリ パスを作成します。

```
/etc/docker/certs.d/IP-address-or-FQDN-of-harbor/
```

例：

```
mkdir /etc/docker/certs.d/10.179.145.77
```

6 `ca.crt` をこのディレクトリに移動します。

例：

```
 mv ca.crt /etc/docker/certs.d/10.179.145.77/ca.crt
```

7 Docker デーモンを再起動します。

```
sudo systemctl restart docker.service
```

8 Docker クライアントを使用して、組み込みの Harbor レジストリにログインします。

```
docker login https://10.179.145.77
```

次のメッセージが表示されます。

```
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
```

プライベート Harbor レジストリへの標準パッケージのプッシュ

パブリック コンテナ レジストリから、スーパーバイザー サービスとして実行されているプライベート Harbor レジストリに標準パッケージをプッシュするには、このトピックを参照してください。

前提条件

次の前提条件を満たすようにします。

- Harbor スーパーバイザー サービスのインストール。
- Ubuntu ホストを Docker クライアントとして構成します。Harbor レジストリ証明書を使用した Docker クライアントの構成を参照してください。
- Ubuntu ホストに jq をインストールします。https://jqlang.github.io/jq/download/を参照してください。

Carvel イメージ パッケージ ユーティリティのインストール

kubectl がインストールされている Ubuntu クライアントに Carvel `imgpkg` ユーティリティをインストールします。

1 `imgpkg` をインストールします。

```
wget -O- https://carvel.dev/install.sh > install.sh
sudo bash install.sh
```

2 インストールを確認します。

```
imgpkg version
```

予期される結果：

```
imgpkg version 0.41.1
```

各標準パッケージで使用可能なイメージの一覧表示

Cert Manager

```
imgpkg tag list -i projects.registry.vmware.com/tkg/packages/standard/cert-manager
```

Envoy を使用する Contour

```
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
```

Alertmanager を使用する Prometheus

```
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
```

パブリック VMware レジストリからの標準パッケージ イメージのプル

パブリック VMware レジストリ (https://projects.registry.vmware.com/) から標準の Tanzu パッケージをプルします。プルするバージョンに合わせてバージョンを更新します。

Cert Manager

```
docker pull projects.registry.vmware.com/tkg/packages/standard/cert-manager:v1.7.2_vmware.3-tkg.3
```

Envoy を使用する Contour

```
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
```

Alertmanager を使用する Prometheus

```
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
```

プライベート Harbor レジストリでのプロジェクトの作成

Tanzu パッケージをホストするためのパブリック プロジェクトを Harbor に作成します。

1 プライベート Harbor レジストリにログインします。

2 Harbor で、[プロジェクト] - [新しいプロジェクト] の順に選択します。

3 [tanzu-packages] という名前の新しいパブリック プロジェクトを作成します。

標準パッケージ イメージのタグ付け

次の構文を使用してイメージをタグ付けします。

```
docker tag SOURCE_IMAGE[:TAG] harbordomain.com/tanzu-packages/REPOSITORY[:TAG]
```

ここで、

- SOURCE_IMAGE[:TAG] はプルしたイメージの名前です
- *harbordomain.com* は Harbor サーバの DNS 名です
- REPOSITORY[:TAG] はイメージ タグの名前です

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
```

Envoy を使用する Contour

```
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
```

Alertmanager を使用する Prometheus

```
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
```

プライベート Harbor レジストリへの標準パッケージ イメージのプッシュ

次の構文を使用してイメージをプッシュします。

```
docker push harbordomain.com/tanzu-packages/PACKAGE
```

ここで、

- *harbordomain.com* は Harbor サーバの DNS 名です
- *tanzu-packages* は Harbor プロジェクトの名前です。
- *PACKAGE* は Tanzu パッケージの名前です
- *vX.X.X* はパッケージのタグ バージョンです

Cert Manager

```
docker push harbordomain.com/tanzu-packages/cert-manager:v1.7.2
```

Envoy を使用する Contour

```
docker push harbordomain.com/tanzu-packages/contour:v1.23.5
```

ExternalDNS

```
docker push harbordomain.com/tanzu-packages/external-dns:v0.12.2
```

Alertmanager を使用する Prometheus

```
docker push harbordomain.com/tanzu-packages/prometheus:v2.37.0
```

Grafana

```
docker push harbordomain.com/tanzu-packages/grafana:v7.5.17
```

Fluent Bit

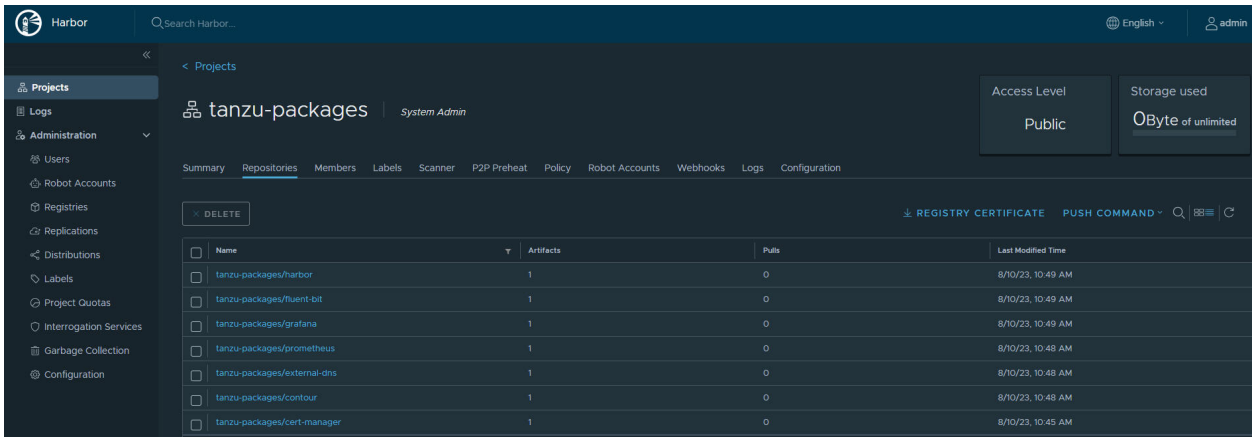
```
docker push harbordomain.com/tanzu-packages/fluent-bit:v1.9.5
```

Harbor

```
docker push harbordomain.com/tanzu-packages/harbor:v2.7.1
```

イメージをプライベート Harbor レジストリにプッシュしたら、Harbor Web インターフェイスに各イメージが表示されることを確認します。

図 14-1. プライベート Harbor レジストリ内の標準の Tanzu パッケージ



デプロイ用のコンテナ イメージのプル

イメージがプライベート Harbor レジストリ内にあることを確認するには、次の構文を使用してレジストリからイメージをプルします。

```
docker pull harbordomain.com/tanzu-packages/PACKAGE:TAG
```

例：

```
docker pull harbordomain.com/tanzu-packages/fluent-bit:v1.9.5
```

cURL を使用して Harbor API を呼び出し、パッケージを一覧表示できます。そのためには、Harbor Web インターフェイスから Harbor 証明書をダウンロードして、次のコマンドを実行します。

```
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'
```

このコマンドは、使用可能な tanzu-packages を返します。

```
"tanzu-packages/harbor"
"tanzu-packages/fluent-bit"
"tanzu-packages/grafana"
"tanzu-packages/prometheus"
"tanzu-packages/external-dns"
"tanzu-packages/contour"
"tanzu-packages/cert-manager"
```

TKG サービス クラスタでのスナップショットの作成

15

TKG サービス クラスタでは、ボリュームのスナップショットおよびリストア機能がサポートされます。DevOps ユーザーは、ボリューム スナップショットを作成して TKG クラスタ内のワークロードを保護できます。

スナップショットを使用すると、スナップショット データがあらかじめ設定された新しいボリュームをプロビジョニングできます。

前提条件

TKG サービス クラスタの CSI スナップショットを作成するには、環境が次の前提条件を満たしている必要があります。

- vSphere 8.0 Update 2 以降。
- CSI スナップショットをサポートする Tanzu Kubernetes リリース。vSphere 8.0.2 以降の場合は、TKr v1.26.5 以降です。「VMware Tanzu Kubernetes リリースのリリース ノート」を参照してください。
- 互換性のある最新の スーパーバイザー バージョン。「VMware vSphere with Tanzu 8.0 リリース ノート」を参照してください。

要件

CSI スナップショット機能は TKG パッケージとして提供されます。CSI スナップショット パッケージを使用するための要件は次のとおりです。

- TKG 標準パッケージ リポジトリ バージョン v2023.9.19 以降を使用します。Tanzu Standard パッケージ リポジトリのバージョンを参照してください。
- Cert Manager パッケージをインストールします。「Installing and Using VMware Tanzu Packages」を参照してください。
- Tanzu CLI を使用して、vsphere-pv-csi-webhook をインストールおよびデプロイします。vSphere PVCSI Webhook のインストールとデプロイを参照してください。
- Tanzu CLI を使用して external-csi-snapshot-webhook をインストールして展開します。外部 CSI スナップショット Webhook のインストールとデプロイを参照してください。

ガイドラインと制限事項

TKG クラスタでスナップショットおよびリストア機能を使用する場合は、次のガイドラインに従ってください。

- ボリュームのスナップショット操作とリストア操作をサポートするのは、ブロック ボリュームのみです。これらの操作を vSphere ファイル ボリュームで使用することはできません。
- VolumeSnapshot から PVC を作成する場合は、元の VolumeSnapshot と同じデータストアに配置する必要があります。そうしなかった場合、その PVC のプロビジョニングは失敗し、次のエラーが表示されます。

  ```
  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]>
  ```

  VolumeSnapshot から作成するターゲット PVC のデータストアは、PVC 定義内の StorageClass によって決まります。ターゲット PVC の StorageClass と元のソース PVC の StorageClass が同じデータストア（ソース PVC のデータストア）を参照していることを確認してください。このルールは、StorageClass 定義のトポロジ要件にも適用されます。この要件では、同じ共通データストアを参照する必要もあります。トポロジ要件が競合すると、上記と同じエラーが発生します。
- 関連付けられたスナップショットを含むボリュームを削除または拡張することはできません。ソース ボリュームを拡張または削除するには、すべてのスナップショットを削除してください。
- スナップショットからボリュームを作成する場合は、ボリュームのサイズがスナップショットのサイズと一致していることを確認します。
- スナップショットでは、ストレージ割り当ての監視はサポートされていません。
- vSphere 構成では、ボリュームあたりのスナップショットの最大数を構成できません。パフォーマンスを向上させるには、仮想ディスクごとに 2 ～ 3 個のスナップショットを使用します。詳細については、「Best practices for using VMware snapshots in the vSphere environment」を参照してください。

  vSAN ESA の場合は、ボリュームごとに最大 32 個のスナップショットを使用します。vSAN ESA の詳細については、「vSAN Express Storage Architecture」を参照してください。

次のトピックを参照してください。

- 外部 CSI スナップショット Webhook のインストールとデプロイ
- vSphere PVCSI Webhook のインストールとデプロイ
- TKG サービス クラスタでのスナップショットの作成

外部 CSI スナップショット Webhook のインストールとデプロイ

TKG サービス クラスタでスナップショット テクノロジーを使用できるようにするには、TKG クラスタに外部 CSI スナップショット Webhook をインストールしてデプロイする必要があります。外部 CSI スナップショット Webhook は、アドミッション要求に応答する HTTP コールバックを備えたオープンソース コンポーネントです。ボリューム スナップショット オブジェクトの検証を行うことができます。

外部 CSI スナップショット Webhook は、スーパーバイザーに自動的にインストールされます。このトピックは、TKG サービス クラスタにのみ適用されます。

前提条件

- 実行中のスーパーバイザーが必要です。
- Tanzu CLI と kubectl をインストールしておく必要があります。詳細については、『TKG サービス クラスタ用 CLI ツールのインストール』を参照してください。
- TKG クラスタにログインしておく必要があります。詳細については、『vCenter SSO 認証を使用した TKG サービス クラスタへの接続』を参照してください。

外部 CSI スナップショット Webhook をインストールするための TKG サービス クラスタの準備

次の手順に従って、TKG サービス クラスタに外部 CSI スナップショット Webhook をインストールします。

手順

1 外部 CSI スナップショット Webhook をデプロイする TKG クラスタの管理者認証情報を取得します。

```
tanzu cluster kubeconfig get my-cluster --admin
```

2 ターゲット TKG クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context my-cluster-admin@my-cluster
```

3 パッケージ リポジトリ（tanzu-standard リポジトリなど）がクラスタにない場合は、パッケージ リポジトリをインストールします。

ターゲット クラスタがプランベースのレガシー クラスタの場合は、この手順をスキップできます。プランベースのクラスタの場合、tanzu-package-repo-global 名前空間で tanzu-standard パッケージ リポジトリが自動的に有効になります。

```
tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --namespace tkg-system
```

- PACKAGE-REPO-NAME は、パッケージ リポジトリの名前（tanzu-standard など）または ADDITIONAL_IMAGE_REGISTRY 変数で構成されたプライベート イメージ レジストリの名前です。
- PACKAGE-REPO-ENDPOINT は、パッケージ リポジトリの URL です。

4 cert-manager をまだインストールしていない場合は、インストールします。

詳細については、『Cert Manager のインストール』を参照してください。

結果

これで、外部 CSI スナップショット Webhook をデプロイできます。

外部 CSI スナップショット Webhook のデプロイ

次の手順に従って、TKG サービス クラスタに外部 CSI スナップショット Webhook をデプロイします。

手順

1 外部 CSI スナップショット Webhook パッケージがクラスタで使用可能であることを確認します。

```
tanzu package available list -A
```

パッケージが使用できない場合は、必要な外部 CSI スナップショット Webhook パッケージを含むパッケージ リポジトリが正しくインストールされていることを確認します。手順については、「外部 CSI スナップショット Webhook をインストールするための TKG サービス クラスタの準備」の手順 3 を参照してください。

2 使用可能なパッケージのバージョンを取得します。

```
tanzu package available list external-csi-snapshot-webhook.tanzu.vmware.com -A
```

3 使用可能な適切なバージョンのパッケージをインストールします。

```
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 は、手順 2 で取得したパッケージ バージョンを指定します。

4 外部 CSI スナップショット Webhook パッケージがインストールされたことを確認します。

```
tanzu package installed list -A
```

パッケージの詳細を表示するには、次のコマンドを実行することもできます。

```
tanzu package installed get external-csi-snapshot-webhook --namespace kube-system
```

5 external-csi-snapshot-webhook アプリケーションが TARGET-NAMESPACE で正常に調整されたことを確認します。

```
kubectl get apps -A
```

ステータスが `Reconcile Succeeded` でない場合は、external-csi-snapshot-webhook アプリケーションの完全なステータスの詳細を表示します。完全なステータスを表示すると、問題のトラブルシューティングに役立ちます。

```
kubectl get app external-csi-snapshot-webhook --namespace kube-system -o yaml
```

トラブルシューティングを実行しても問題を解決できない場合は、次のコマンドを使用してパッケージをアンインストールしてから、再度インストールしてください。

```
tanzu package installed delete external-csi-snapshot-webhook --namespace kube-system
```

6 クラスタ内のすべてのポッドを一覧表示して、external-csi-snapshot-webhook が実行されていることを確認します。

```
kubectl get pods -A
```

kube-system 名前空間に external-csi-snapshot-webhook ポッドが作成されていることを確認します。

vSphere PVCSI Webhook のインストールとデプロイ

TKG サービス クラスタに vSphere PVCSI Webhook をインストールしてデプロイします。vSphere PVCSI Webhook は、CSI アドミッション要求に応答するコールバックを備えたコンポーネントです。Kubernetes オブジェクト（パーシステント ボリューム要求、パーシステント ボリューム、ストレージ クラスなど）の検証を行うことができます。

vSphere PVCSI Webhook は、スーパーバイザーに自動的にインストールされます。このトピックは、TKG サービス クラスタにのみ適用されます。

前提条件

- 実行中のスーパーバイザーが必要です。
- Tanzu CLI と kubectl をインストールしておく必要があります。詳細については、『TKG サービス クラスタ用 CLI ツールのインストール』を参照してください。
- TKG クラスタにログインしておく必要があります。詳細については、『vCenter SSO 認証を使用した TKG サービス クラスタへの接続』を参照してください。

vSphere PVCSI Webhook をインストールするための TKG サービス クラスタの準備

次の手順に従って、vSphere PVCSI Webhook をインストールするための TKG サービス クラスタの準備を行います。

手順

1 vSphere PVCSI Webhook をデプロイする TKG クラスタの管理者認証情報を取得します。

```
tanzu cluster kubeconfig get my-cluster --admin
```

2 ターゲット TKG クラスタがプロビジョニングされている vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context my-cluster-admin@my-cluster
```

3 vsphere-pv-csi-webhook パッケージを含むパッケージ リポジトリ（tanzu-standard リポジトリなど）がクラスタにインストールされていない場合は、パッケージ リポジトリをインストールします。

ターゲット クラスタがプランベースのレガシー クラスタの場合は、この手順をスキップできます。プランベースのクラスタの場合、tanzu-package-repo-global 名前空間で tanzu-standard パッケージ リポジトリが自動的に有効になります。

```
tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --namespace tkg-system
```

- PACKAGE-REPO-NAME は、パッケージ リポジトリの名前（tanzu-standard など）または ADDITIONAL_IMAGE_REGISTRY 変数で構成されたプライベート イメージ レジストリの名前です。
- PACKAGE-REPO-ENDPOINT は、パッケージ リポジトリの URL です。

4 cert-manager をまだインストールしていない場合は、インストールします。

詳細については、『Cert Manager のインストール』を参照してください。

結果

これで、vSphere PVCSI Webhook をデプロイできます。

vSphere PVCSI Webhook のデプロイ

次の手順に従って、TKG サービス クラスタに vSphere PVCSI Webhook をデプロイします。

手順

1 vSphere PVCSI Webhook パッケージがクラスタで使用可能であることを確認します。

```
tanzu package available list -A
```

パッケージが使用できない場合は、必要な vSphere PVCSI Webhook パッケージを含むパッケージ リポジトリが正しくインストールされていることを確認します。手順については、「vSphere PVCSI Webhook をインストールするための TKG サービス クラスタの準備」の手順 3 を参照してください。

2 使用可能なパッケージのバージョンを取得します。

```
tanzu package available list vsphere-pv-csi-webhook.tanzu.vmware.com -A
```

3 使用可能な適切なバージョンのパッケージをインストールします。

```
tanzu package install vsphere-pv-csi-webhook --package vsphere-pv-csi-webhook.tanzu.vmware.com --version AVAILABLE-PACKAGE-VERSION --namespace TARGET-NAMESPACE
```

- TARGET-NAMESPACE は、vsphere-pv-csi-webhook パッケージをインストールする名前空間を指定します。

  注： TARGET-NAMESPACE は、vsphere-pv-csi パッケージがインストールされている名前空間と同じである必要があります。

`--namespace` フラグを指定しない場合、Tanzu CLI はパッケージとそのリソースをデフォルトの名前空間（例：vsphere-pv-csi-webhook パッケージの場合は `vmware-system-csi`）にインストールします。たとえば、`kubectl create namespace vmware-system-csi` を実行する場合は、指定された名前空間がすでに存在している必要があります。

- AVAILABLE-PACKAGE-VERSION は、手順 2 で取得したパッケージ バージョンを指定します。

4 vSphere PVCSI Webhook パッケージがインストールされていることを確認します。

```
tanzu package installed list -A
```

パッケージの詳細を表示するには、次のコマンドを実行することもできます。

```
tanzu package installed get vsphere-pv-csi-webhook --namespace TARGET-NAMESPACE
```

5 vsphere-pv-csi-webhook アプリケーションが TARGET-NAMESPACE で正常に調整されたことを確認します。

```
kubectl get apps -A
```

ステータスが `Reconcile Succeeded` でない場合は、vsphere-pv-csi-webhook アプリケーションの完全なステータスの詳細を表示します。完全なステータスを表示すると、問題のトラブルシューティングに役立ちます。

```
kubectl get app vsphere-pv-csi-webhook --namespace TARGET-NAMESPACE -o yaml
```

トラブルシューティングを実行しても問題を解決できない場合は、次のコマンドを使用してパッケージをアンインストールしてから、再度インストールしてください。

```
tanzu package installed delete vsphere-pv-csi-webhook --namespace TARGET-NAMESPACE
```

6 クラスタ内のすべてのポッドを一覧表示して、vsphere-pv-csi-webhook が実行されていることを確認します。

```
kubectl get pods -A
```

vmware-system-csi または TARGET-NAMESPACE に vsphere-pv-csi-webhook ポッドが作成されていることを確認します。

TKG サービス クラスタでのスナップショットの作成

TKG サービス クラスタでは、スナップショットを動的にプロビジョニングしたり、ブロック ボリュームの事前プロビジョニング済みボリューム スナップショットを作成したりできます。既存のスナップショットをリストアすることもできます。

注： ボリューム スナップショット クラスを作成し、それらを使用してボリューム スナップショットを作成することはできません。既存のボリューム スナップショット クラスのみを使用してください。既存のボリューム スナップショット クラスでは、Delete deletionPolicy のみがサポートされます。Retain deletionPolicy を使用したボリューム スナップショット クラスの作成はサポートされていません。

TKG サービス クラスタでの動的にプロビジョニングされたスナップショットの作成

TKG サービス クラスタでスナップショットを動的にプロビジョニングします。

手順

1 StorageClass が存在することを確認します。

```
$ 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 手順 1 の StorageClass を使用して PVC を作成します。

例として、次の YAML を使用します。

```
$ 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 VolumeSnapshotClass が使用可能であることを確認します。

```
$ kubectl get volumesnapshotclass
NAME                         DRIVER                    DELETIONPOLICY   AGE
volumesnapshotclass-delete   csi.vsphere.vmware.com    Delete           11d
```

4 手順 3 で取得した VolumeSnapshotClass を使用して VolumeSnapshot を作成します。

```
$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
```

TKG サービス クラスタでの事前プロビジョニング済みスナップショットの作成

スーパーバイザー から同じブロック ボリューム (PVC) のボリューム スナップショットを使用して、TKG サービス クラスタに任意のブロック ボリューム (PVC) の事前プロビジョニング済みボリューム スナップショットを作成します。

次の手順に従って、スーパーバイザー 内の残りの基盤となるスナップショットの情報を使用して新しい TKG クラスタ内にボリューム スナップショットを静的に作成します。

注： スーパーバイザー でボリューム スナップショットを直接作成することはできません。

前提条件

- Kubernetes スナップショットの作成に関する情報を理解しておく必要があります。詳細については、Kubernetes ドキュメントの「Volume Snapshots」ページを参照してください。
- ボリューム スナップショットが次の条件を満たしていることを確認します。
  - ソース PVC が配置されている名前空間内にボリューム スナップショットが存在する。
  - TKG クラスタが配置されている名前空間内にボリューム スナップショットが存在する。

同じ名前空間内の別の TKG クラスタで不要になったボリューム スナップショットを新しい TKG クラスタで再利用することもできます。そのためには、元の TKG クラスタ内の `VolumeSnapshotContent` の `deletionPolicy` を `Retain` に変更してから、対応する `VolumeSnapshot` オブジェクトと `VolumeSnapshotContent` オブジェクトを削除します。

手順

1 スーパーバイザー 内の元の VolumeSnapshot オブジェクトの名前を書き留めます。

古い TKG クラスタのボリューム スナップショットを再利用する場合は、古い TKG クラスタ内の古い VolumeSnapshotContent オブジェクトの snapshotHandle から スーパーバイザー の VolumeSnapshot 名を取得することもできます。

2 VolumeSnapshotContent オブジェクトを作成します。

YAML ファイルで、次の項目の値を指定します。

snapshotHandle には、手順 1 で取得した スーパーバイザー の VolumeSnapshot 名を入力します。

注： 注：別の TKG クラスタのボリューム スナップショットを再利用する場合は、新しい TKG クラスタに VolumeSnapshotContent を作成する前に、deletionPolicy が Retain に設定されている古い TKG クラスタから VolumeSnapshot オブジェクトと VolumeSnapshotContent オブジェクトを削除します。

例として、次の YAML マニフェストを使用します。

```
--------
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 手順 2 で作成した VolumeSnapshotContent オブジェクトと一致する VolumeSnapshot を作成します。

```
---------
apiVersion: snapshot.storage.k8s.io/vl
kind: VolumeSnapshot
metadata:
 name: static-tkg-block-snapshot
spec:
 source:
  volumeSnapshotContentName: static-tkg-block-snapshotcontent
---------
```

4 手順 3 で作成した VolumeSnapshot の ReadyToUse が「true」になっていることを確認します。

```
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
```

TKG サービス クラスタでのボリューム スナップショットのリストア

すでに作成されているボリューム スナップショットをリストアします。

手順

1 リストアするボリューム スナップショットが TKG クラスタで使用可能であることを確認します。

```
$ 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 ボリューム スナップショットから PVC を作成します。

```
$ 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
```

TKG サービス クラスタのストレージの管理

16

このセクションでは、TKG サービス クラスタのストレージの管理について説明します。

次のトピックを参照してください。

- TKG サービス クラスタのストレージの概念
- ノード ボリューム マウントの使用に関する考慮事項
- TKG サービス クラスタの vSphere ストレージ ポリシーの作成
- TKG サービス クラスタでのステートフル アプリケーションの動的パーシステント ボリュームのプロビジョニング
- TKG サービス クラスタでの静的パーシステント ボリュームのプロビジョニング
- TKG サービス クラスタのパーシステント ボリュームの拡張

TKG サービス クラスタのストレージの概念

TKG クラスタ ワークロードでは、パーシステント ストレージが必要になることがあります。TKG サービス クラスタの vSphere ストレージの概念と考慮事項については、このトピックの情報を参照してください。

TKG サービス クラスタの vSphere ストレージ ポリシー

TKG サービス クラスタにパーシステント ストレージ リソースを提供するために、vSphere 管理者は、さまざまなストレージ要件を記述する vSphere ストレージ ポリシーを構成します。次に、1 つ以上のストレージ ポリシーを、TKG クラスタがデプロイされる vSphere 名前空間に追加します。vSphere 名前空間に割り当てられたストレージ ポリシーでは、TKG クラスタ ノードとワークロードを vSphere ストレージ環境に配置する方法、および TKG クラスタからアクセスしてパーシステント ストレージに使用できるデータストアが決定されます。

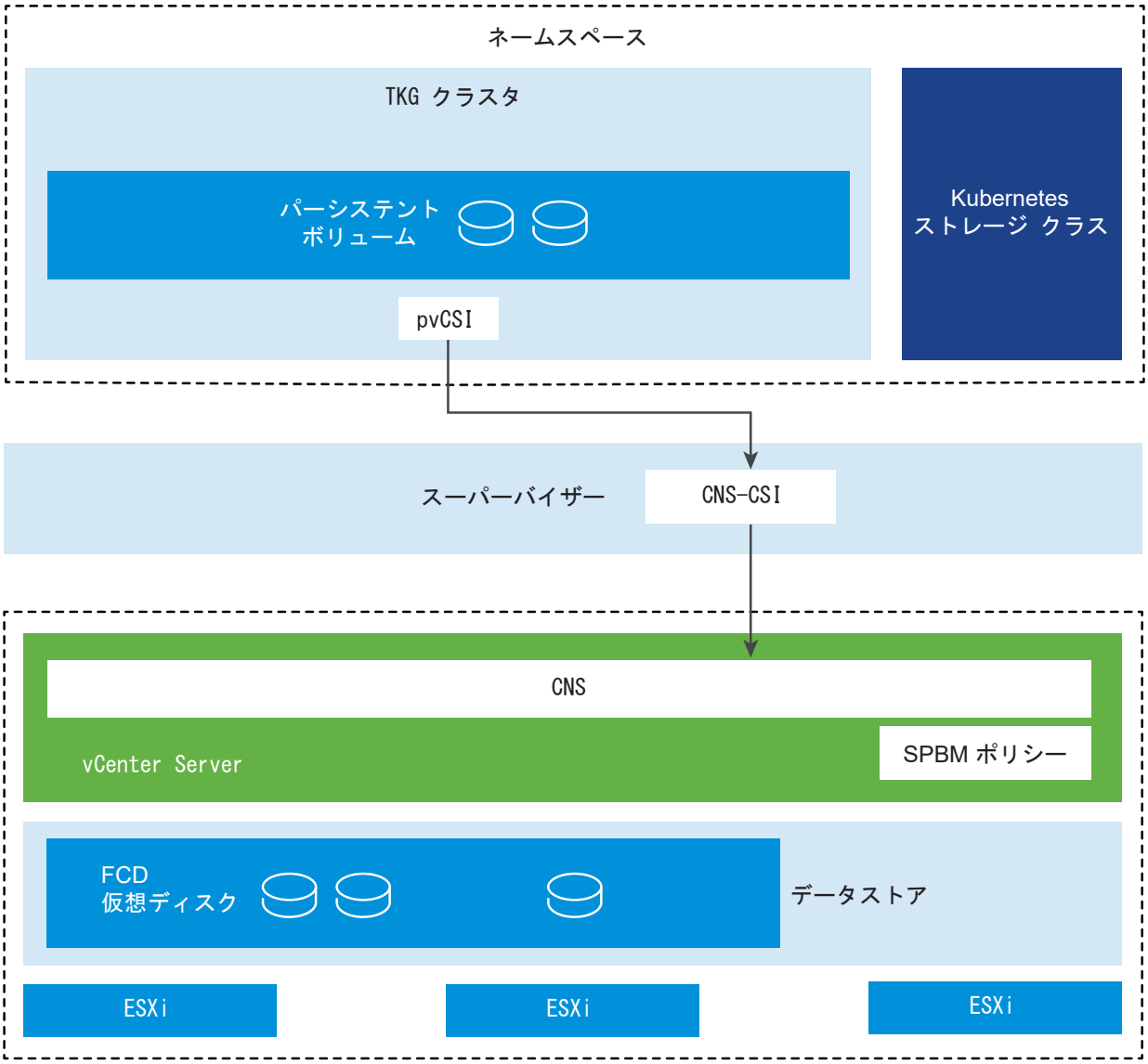
vSphere ストレージ ポリシーが vSphere 名前空間に割り当てられると、その vSphere 名前空間に対応する Kubernetes ストレージ クラスがシステムによって作成されます。対応する Kubernetes ストレージ クラスは、その vSphere 名前空間でプロビジョニングされた TKG クラスタに伝達されます。

TKG クラスタでは、各ストレージ クラスに 2 つのエディション（`Immediate` および `WaitForFirstConsumer` バインド モード）があります。選択するエディションは、要件によって異なります。TKG サービス クラスタのストレージ クラス エディションを参照してください。

TKG サービス クラスタと vSphere ストレージの統合方法

スーパーバイザーと vSphere ストレージを統合するために、TKG クラスタでは準仮想化 CSI (pvCSI) が使用されます。

pvCSI は TKG クラスタ用に変更された vSphere CNS-CSI ドライバのバージョンです。pvCSI は TKG クラスタに配置され、TKG クラスタから送信されるすべてのストレージ関連の要求に対処します。要求は CNS-CSI に配信され、vCenter Server の CNS に伝達されます。その結果、pvCSI は CNS コンポーネントとの直接通信は行わず、すべてのストレージ プロビジョニング操作に CNS-CSI を使用します。CNS-CSI とは異なり、pvCSI はインフラストラクチャの認証情報を必要としません。vSphere 名前空間 のサービス アカウントを使用して構成されます。

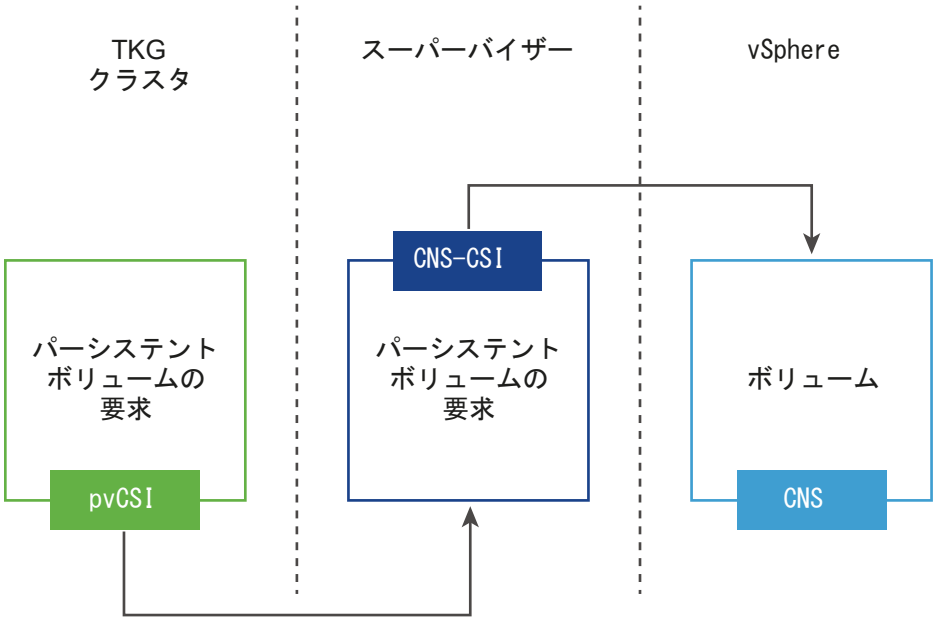


パーシステント ボリュームの作成方法

次の図は、TKG クラスタ内のストレージ関連の操作（パーシステント ボリューム要求 (PVC) の作成など）に対して、さまざまなコンポーネントがどのように相互作用するかを示しています。

DevOps エンジニアは TKG クラスタで kubectl を使用して PVC を作成します。このアクションにより、スーパーバイザーで一致する PVC が生成され、CNS-CSI がトリガされ、CNS ボリューム作成 API が呼び出されます。

ボリュームの作成が正常に完了すると、操作はスーパーバイザーを介して元の TKG クラスタに伝達されます。クラスタ ユーザーは、バインド状態のパーシステント ボリュームとパーシステント ボリューム要求をスーパーバイザーで確認できます。また、バインド状態のパーシステント ボリュームとパーシステント ボリューム要求は TKG クラスタでも確認できます。



ノード ボリューム マウントの使用に関する考慮事項

1 つ以上のノード ボリューム マウントを指定して TKG サービス クラスタをプロビジョニングできます。これを行う前に、重要な考慮事項に注意してください。

ノード ボリュームのマウントに関する考慮事項

次の表に、TKG サービス クラスタのノード ボリューム マウント ポイントの概要を示します。ノード ボリューム マウントの制限の詳細については、次のナレッジベース記事を参照してください：https://kb.vmware.com/s/article/92153。

クラスタのプロビジョニング後にボリューム マウントを変更する際の詳細については、次を参照してください；kubectl を使用したクラスタの手動スケーリング。

ボリューム マウント	ノード	サポート	降順
/var/lib/containerd	ワーカー	全二重	コンテナ イメージのキャッシュに使用できるサイズを増やします。
/var/lib/kubelet	ワーカー	全二重	短期コンテナ ストレージに使用できるサイズを増やします。

ボリューム マウント	ノード	サポート	降順
/var/lib/etcd	制御プレーン	なし	etcd には 2 GB のハード制限があるため、etcd のサイズは増やしません。また、PVC タイムアウトを原因とするクラスタ作成を回避することもできます。
/（ルート） /var /var/lib /etc	制御プレーン ワーカー	なし	コア システム プロセスで使用されているディレクトリは、リストに含まれているものなども含め、一切サポートされません（リストは網羅的ではありません）。

TKG サービス クラスタの vSphere ストレージ ポリシーの作成

vSphere 名前空間 に割り当てる vSphere ストレージ ポリシーは、Kubernetes ストレージ クラスに変換されます。このストレージ クラスを使用して、TKG クラスタ ノードとパーシステント ボリュームの vSphere データストア内への配置方法を制御します。vSphere Zones の vSphere ストレージ ポリシーは、ゾーン トポロジを構成するすべての vSphere クラスタのストレージと互換性がある必要があります。

単一ゾーン スーパーバイザー の vSphere ストレージ ポリシーの定義

単一ゾーン スーパーバイザー のストレージ クラスを作成する手順を実行します。

1 vSphere Client で、[ポリシーおよびプロファイル] を選択します。

2 [仮想マシン ストレージ ポリシー] - [作成] の順に選択します。

3 [vCenter Server] を選択します。

4 ストレージ ポリシーに *TKG2-cluster-storage-class* などの記述的な名前を付けます。

作成されるストレージ クラスには同じ名前が使用されます。

5 ストレージの [ポリシー構造] で、[「VMFS」ストレージでルールを有効化] を選択します。

6 VMFS ルールでは、[可能な場合は容量を節約] を選択します。

7 ストレージ互換性を確認し、[次へ] をクリックします。

8 [終了] をクリックして、ストレージ ポリシーの作成を完了します。

ストレージ ポリシーを vSphere 名前空間 に割り当てるには、「TKG サービス クラスタ向けの vSphere 名前空間の構成」を参照してください。

3 つのゾーン スーパーバイザー の vSphere ストレージ ポリシーの定義

3 つのゾーンにデプロイされた スーパーバイザー を使用している場合、上記と同じ手順を次の追加手順とともに実行します。

- ストレージの [ポリシー構造] については、[ストレージ トポロジ] - [使用ドメインの有効化] の順に選択します。
- [使用ドメイン] には、ストレージ ポリシー タイプとして [ゾーン] を指定します。

図 16-1. ゾーン ストレージ クラス 1/2

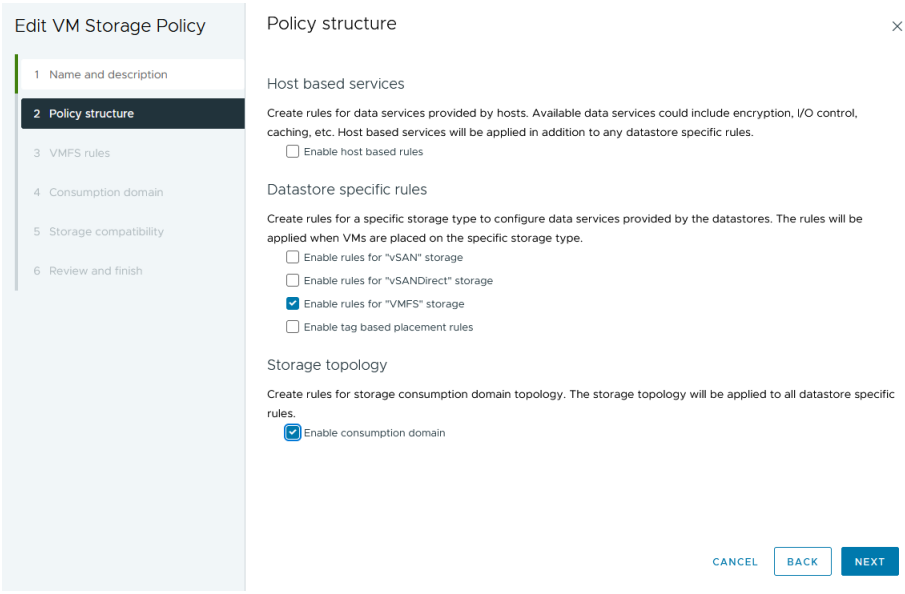
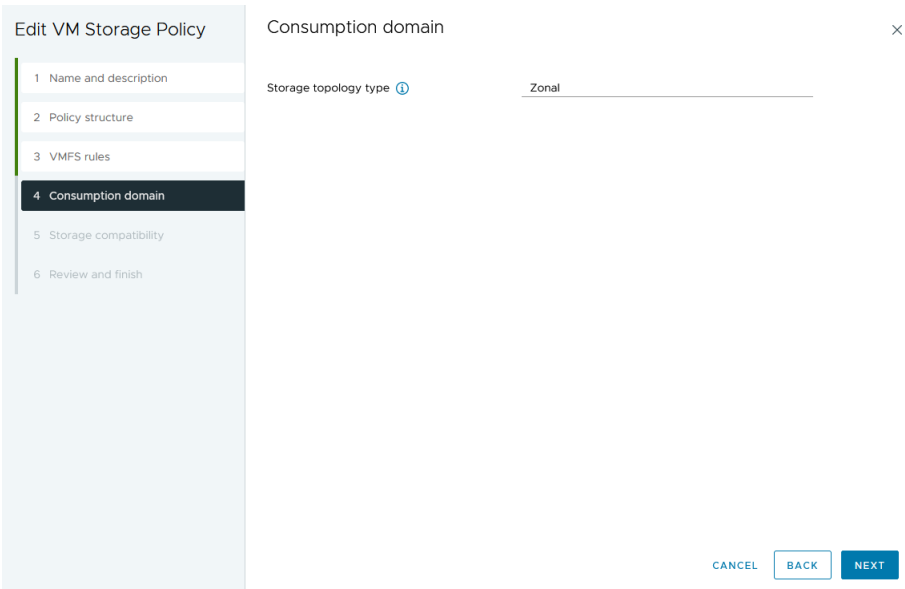


図 16-2. ゾーン ストレージ クラス 2/2



高度な vSphere ストレージ ポリシーの作成

環境によっては、vSAN または VVol のいずれかの追加設定が必要になる場合があります。VMFS および NFS を使用している場合、vSphere ストレージ ポリシーにタグが含まれます。

TKG クラスタでの使用のために vSphere 名前空間 に適用可能な、さらに高度なストレージ ポリシー タイプ（タグベースのストレージなど）を作成するには、vSphere IaaS 制御プレーンのインストールと構成を参照してください。

TKG サービス クラスタでのステートフル アプリケーションの動的パーシステント ボリュームのプロビジョニング

データベースなどのステートフル アプリケーションでは、セッション間でデータが保存されるため、データを格納するためのパーシステント ストレージが必要です。保持されたデータは、アプリケーションの状態と呼ばれます。後でデータを取得し、次のセッションで使用することができます。Kubernetes は、状態とデータを保持できるオブジェクトとしてパーシステント ボリュームを提供します。

vSphere 環境では、パーシステント ボリューム オブジェクトは、データストアにある仮想ディスクによってバッキングされます。データストアはストレージ ポリシーによって表されます。vSphere 管理者が **`gold`** などのストレージ ポリシーを作成して、スーパーバイザー内の名前空間に割り当てると、そのストレージ ポリシーは、vSphere 名前空間および使用可能な TKG クラスタでは、一致する Kubernetes ストレージ クラスとして表示されます。

DevOps エンジニアは、このストレージ クラスをパーシステント ボリュームの要求指定で使用できます。その後、パーシステント ボリュームの要求から得たストレージを使用するアプリケーションをデプロイできます。この例では、アプリケーションのパーシステント ボリュームが動的に作成されています。

前提条件

vSphere 管理者が適切なストレージ ポリシーを作成し、そのポリシーを名前空間に割り当てていることを確認します。

手順

1 vSphere Kubernetes 環境内の名前空間にアクセスします。

2 ストレージ クラスを使用できることを確認します。

3 パーシステント ボリュームの要求を作成します。

a パーシステント ボリュームの要求設定を含む YAML ファイルを作成します。

この例では、ファイルは **gold** ストレージ クラスを参照しています。

ReadWriteMany パーシステント ボリュームをプロビジョニングするには、`accessModes` を `ReadWriteMany` に設定します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gold
  resources:
    requests:
        storage: 3Gi
```

b TKG クラスタにパーシステント ボリュームの要求を適用します。

```
kubectl apply -f pvc_name.yaml
```

このコマンドでは、要求のストレージ要件を満たすバッキング仮想ディスクを持つ Kubernetes パーシステント ボリュームと vSphere ボリュームが動的に作成されます。

c パーシステント ボリュームの要求のステータスを確認します。

```
kubectl get pvc my-pvc
```

出力には、ボリュームがパーシステント ボリュームの要求にバインドされていることが示されます。

```
NAME     STATUS    VOLUME   CAPACITY   ACCESSMODES   STORAGECLASS   AGE
my-pvc   Bound     my-pvc   2Gi        RWO           gold           30s
```

4 パーシステント ボリュームをマウントするポッドを作成します。

a パーシステント ボリュームを含む YAML ファイルを作成します。

このファイルには次のパラメータが含まれます。

```
...
volumes:
    - name: my-pvc
      persistentVolumeClaim:
        claimName: my-pvc
```

b YAML ファイルからポッドをデプロイします。

```
kubectl create -f pv_pod_name.yaml
```

c ポッドが作成されたことを確認します。

```
kubectl get pod
```

```
NAME       READY   STATUS    RESTARTS   AGE
pod_name   1/1     Ready     0          40s
```

結果

設定したポッドでは、パーシステント ボリュームの要求で指定されているパーシステント ストレージが使用されます。

TKG サービス クラスタでの静的パーシステント ボリュームのプロビジョニング

スーパーバイザーから未使用のパーシステント ボリューム要求 (PVC) を使用して、TKG サービス クラスタ内にブロック ボリュームを静的に作成することができます。

PVC は次の条件を満たす必要があります。

- PVC は TKG クラスタが配置されている名前空間内に存在します。
- PVC は、TKG クラスタ内のポッドやスーパーバイザー内の vSphere ポッド にまだ接続されていません。

静的プロビジョニングを使用すると、別の TKG クラスタで不要になった PVC を新しい TKG クラスタで再利用することもできます。再利用するには、元の TKG クラスタ内のパーシステント ボリューム (PV) の `Reclaim policy` を `Retain` に変更して、対応する PV を削除します。

次の手順に従って、残りの基盤となるボリュームの情報を使用して新しい TKG クラスタ内に PVC を静的に作成します。

手順

1 スーパーバイザー 内の元の PVC の名前を書き留めます。

古い TKG クラスタの PVC を再利用する場合は、TKG クラスタ内の古い PV オブジェクトの `volumeHandle` から PVC 名を取得できます。

2 PV を作成します。

YAML ファイルで、次のアイテムの値を指定します。

- `storageClassName` には、スーパーバイザー で PVC が使用しているストレージ クラスの名前を入力できます。
- `volumeHandle` には、取得した PVC 名を入力します。

別の TKG クラスタのボリュームを再利用する場合は、新しい TKG クラスタで PV を作成する前に、古い TKG クラスタから PVC と PV オブジェクトを削除します。

例として、次の YAML マニフェストを使用します。

```
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 作成した PV オブジェクトと一致する PVC を作成します。

`storageClassName` を PV と同じ値に設定します。

```
   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 作成した PV に PVC がバインドされていることを確認します。

```
$ 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
```

TKG サービス クラスタのパーシステント ボリュームの拡張

Kubernetes ボリューム拡張機能を使用して、パーシステント ブロック ボリュームを作成後に拡張できます。TKG サービス クラスタは、オフラインおよびオンラインでのボリューム拡張をサポートしています。

パーシステント ボリュームの拡張について

デフォルトでは、TKG クラスタ環境に表示されるストレージ クラスについて、`allowVolumeExpansion` が `true` に設定されています。このパラメータを使用して、オフライン ボリュームまたはオンライン ボリュームのサイズを変更できます。

ボリュームは、ノードまたはポッドに接続されていない場合はオフラインと見なされます。オンライン ボリュームは、ノードまたはポッドで使用可能なボリュームです。

ボリューム拡張機能のサポート レベルは、vSphere のバージョンによって異なります。拡張をサポートする適切なバージョンに vSphere 環境をアップグレードすると、以前のバージョンの vSphere で作成されたボリュームを拡張できます。

注： 拡張できるのは、パーシステント ブロック ボリュームのみです。現在、vSphere IaaS control plane は、ReadWriteMany ボリュームのボリューム拡張をサポートしていません。

パーシステント ブロック ボリュームを拡張する場合は、次の点に注意してください。

- ボリュームは、ストレージ割り当てで指定された制限まで拡張できます。vSphere IaaS control plane は、パーシステント ボリュームの要求オブジェクトに対する連続したサイズ変更要求をサポートします。
- VMFS、vSAN、vSAN Direct、vVols、NFS など、すべてのタイプのデータストアでボリューム拡張がサポートされます。
- デプロイまたはスタンドアローン ポッドのボリューム拡張を実行できます。
- 静的にプロビジョニングされたボリュームにストレージ クラスが関連付けられている場合は、Tanzu Kubernetes Grid クラスタでそのボリュームのサイズを変更できます。

- StatefulSet の一部として作成されたボリュームは拡張できません。
- ボリュームをバッキングする仮想ディスクにスナップショットがある場合、そのサイズは変更できません。
- vSphere IaaS control plane では、ツリー内または移行後のボリュームの拡張はサポートされません。

オンライン モードでのパーシステント ボリュームの拡張

オンライン ボリュームは、ノードまたはポッドで使用可能なボリュームです。DevOps エンジニアは、オンラインのパーシステント ブロック ボリュームを拡張できます。Tanzu Kubernetes Grid クラスタは、オンラインでのボリューム拡張をサポートしています。

1 次のコマンドを使用して、サイズ変更するパーシステント ボリュームの要求を探します。

この例では、ボリュームで使用されているストレージのサイズは 1 Gi です。

```
$ 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 PVC にパッチを適用してサイズを増やします。たとえば、サイズを 2 Gi に増やします。

この操作により、PVC に関連付けられているボリュームで拡張がトリガされます。

```
$ kubectl patch pvc block-pvc -p '{"spec": {"resources": {"requests": {"storage":
"2Gi"}}}}'
persistentvolumeclaim/block-pvc edited
```

3 PVC と PV の両方のサイズが増加したことを確認します。

```
$ 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 vSphere Client を使用して、新しいパーシステント ボリューム サイズを確認します。

Tanzu Kubernetes Grid クラスタでのボリュームの健全性の監視を参照してください。

オフライン モードでのパーシステント ボリュームの拡張

ボリュームは、ノードまたはポッドに接続されていない場合はオフラインと見なされます。Tanzu Kubernetes Grid クラスタは、オフラインでのボリューム拡張をサポートしています。

1 既存のストレージ クラスに対するパーシステント ボリュームの要求 (PVC) を作成します。

この例では、要求されるストレージのサイズは 1 Gi です。

```
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 PVC にパッチを適用してサイズを増やします。

PVC がノードに接続されていない場合、またはポッドで使用されていない場合は、次のコマンドを使用して PVC にパッチを適用します。この例では、要求されるストレージの増加は 2 Gi です。

この操作により、PVC に関連付けられているボリュームで拡張がトリガされます。

```
kubectl patch pvc example-block-pvc -p '{"spec": {"resources": {"requests": {"storage":
"2Gi"}}}}'
```

3 ボリュームのサイズが増加したことを確認します。

```
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 PVC のサイズ変更が保留中であることを確認します。

注： PVC がポッドによって使用されるまで、PVC のサイズは変更されません。

次の例は、PVC サイズがポッドで使用されていないため変化していないことを示しています。`kubectl describe pvc` を実行すると、PVC に `FilesystemResizePending` 条件が適用されます。ポッドで使用されると、そのサイズが変化します。

```
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 PVC を使用するポッドを作成します。

PVC がポッドで使用されると、ファイルシステムが拡張されます。

6 PVC のサイズが変更されたことを確認します。

```
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
```

`FilesystemResizePending` 条件は PVC から削除されています。ボリュームの拡張が完了しました。

7 vSphere Client を使用して、新しいパーシステント ボリューム サイズを確認します。

Tanzu Kubernetes Grid クラスタでのボリュームの健全性の監視を参照してください。

TKG サービス クラスタのネットワークの管理

17

このセクションでは、TKG サービス クラスタのネットワークの管理について説明します。

次のトピックを参照してください。

- NSX 管理プロキシ サービスのインストール
- TKG サービス クラスタに対する Antrea-NSX Adapter アダプタの有効化
- Tanzu Kubernetes クラスタのデフォルト CNI の設定
- TKG クラスタの TKG サービス構成のカスタマイズ
- TKG クラスタの NSX ネットワーク オブジェクト

NSX 管理プロキシ サービスのインストール

NSX 管理プロキシを Antrea-NSX Adapter と連携して、Antrea ベースの TKG サービス クラスタから NSX Manager にアクセスできます。スーパーバイザー管理ネットワークとワークロード ネットワーク間に隔離がある場合は、NSX 管理プロキシが必要です。プロキシは、NSX 管理プロキシ サービスのデプロイ後に作成されたすべての TKGS クラスタに適用されます。

使用事例

NSX 管理プロキシは、TKG サービス クラスタに対する Antrea-NSX Adapter アダプタの有効化 と組み合わせて使用することを目的としています。

管理ネットワークとワークロード ネットワークの間に隔離がある状態でスーパーバイザーがデプロイされている場合 (ほとんどはこれに該当)、Antrea-NSX Adapter が構成された TKGS クラスタから NSX 管理プレーンにアクセスするには、NSX 管理プロキシを使用する必要があります。システムがこのプロキシを検出すると、スーパーバイザーはプロキシを Antrea-NSX Adapter 構成に自動的に渡します。プロキシがインストールされていない場合は、管理ネットワークが隔離されていると、Antrea-NSX Adapter の起動に失敗します。

NSX 管理プロキシ サービスのデプロイ後に Antrea-NSX Adapter をインストールできます。TKG サービス クラスタに対する Antrea-NSX Adapter アダプタの有効化を参照してください。

前提条件

NSX 管理プロキシは、スーパーバイザー サービスとしてインストールされます。プロキシ サービスをインストールするには、次の前提条件を満たす必要があります。

- vSphere 8 U3 (8.0.3) 以降
- NSX 4.1 以降
- スーパーバイザー が NSX ネットワークで有効になっている
- vCenter Server でのスーパーバイザー サービスの管理権限
- スーパーバイザー サービスに精通している

必要な YAML ファイルのダウンロード

サービス定義とデータ値を含む、必要な YAML ファイルをダウンロードします。

1 スーパーバイザー サービス の配布サイト (https://www.vmware.com/go/supervisor-service) にアクセスします。

2 「NSX Management Proxy」セクションまでスクロールして、次のファイルをダウンロードします。

   a NSX 管理プロキシ サービス定義ファイル：`nsx-management-proxy.yml`

   b NSX 管理プロキシ サービス構成ファイル：`nsx-management-proxy-data-values.yml`

サービスとしての NSX 管理プロキシの登録

NSX 管理プロキシを スーパーバイザー サービス として登録するには、次の手順を実行します。

1 vSphere Client を使用して、[ワークロード管理] - [サービス] の順に移動します。

2 [サービスの新規追加] - [追加] の順に選択します。

3 [アップロード] をクリックします。

4 ダウンロードした `nsx-management-proxy.yml` ファイルを参照して選択します。

5 NSX 管理プロキシ サービス定義が正常にアップロードされていることを確認します。

6 [終了] をクリックします。

7 NSX 管理プロキシ サービスの登録カードが [サービス] タブに表示されていることを確認します。

NSX 管理プロキシ サービスの構成

NSX Manager Proxy サービスをインストールする前に、環境に適した構成値を使用してデータ値ファイルを更新します。

1 テキスト エディタを使用して `nsx-management-proxy-data-values.yml` ファイルを開きます。

2 以下の表に記載されているプロパティを環境に合わせて編集します。

3 変更を保存します。

名前	値
nsxManagers	NSX Manager の IP アドレスのリスト（必須）。 仮想 IP アドレス (VIP) ではなく、実際の IP アドレスを使用する必要があります。NSX 管理クラスタを使用している場合は、3 つの実際の IP アドレスをすべてリストに含める必要があります。
loadBalancerIP	スーパーバイザー ロード バランサの IP アドレス プールの IP アドレス（オプション）。 このフィールドの IP アドレスは、ワークロード ネットワークの「Ingress」CIDR から取得されます。最初のスーパーバイザーの有効化中、または後で vSphere 名前空間を作成してネットワーク設定をオーバーライドするときにワークロード ネットワークが作成されると、IP アドレス CIDR ブロックを受け入れる「Ingress」の構成設定を使用して、そのスーパーバイザー インスタンスで作成されたすべての vSphere 名前空間内でサービス タイプ ロード バランサと Ingress を使用して公開されるサービスに IP アドレスが割り当てられます。 「loadBalancerIP」フィールドが指定されていない場合、システムは「Ingress」CIDR 範囲から使用可能な IP アドレスを自動的に 1 つ割り当てます。 「loadBalancerIP」が指定されている場合は、「Ingress」CIDR 範囲内であり、かつすでに割り当てられている IP アドレスと競合しない IP アドレスを使用する必要があります。 vSphere 名前空間で「kubectl get services -o wide -A」コマンドを使用して、割り当てられた「Ingress」IP アドレスを表示できます。IP アドレスは、kubectl 出力の「EXTERNAL-IP」列にあります。

NSX 管理プロキシ サービスのインストール

次の手順を実行して、NSX 管理プロキシ サービスをインストールします。

1 [ワークロード管理] - [] - [サービス] 画面に移動します。

2 [NSX 管理プロキシ] サービス カードで、[アクション] - [スーパーバイザーへのインストール] の順に選択します。

3 [使用可能] タブを選択します。

4 編集した `nsx-management-proxy-data-values.yml` の内容をコピーして [YAML サービス構成] 入力フィールドに貼り付けます。

5 [OK] をクリックして、Harbor のインストールを続行します。

6 インストールを監視および確認します。

インストールを監視するには、NSX 管理プロキシ サービス カードの スーパーバイザー フィールドを確認します。スーパーバイザー の横にある数字が増えます。目的の状態に達するまで、サービスの状態は [設定中] になります。目的の状態に達すると、サービスの状態は [設定済み] に変化します。

7 NSX 管理プロキシの vSphere 名前空間 があることを確認します。

NSX 管理プロキシがインストールされると、サービス インスタンス用に vSphere 名前空間 が作成されます。

8 プロキシ ロード バランサの IP アドレスを取得します。

プロキシ ロード バランサの IP アドレスは、NSX 管理プロキシの vSphere 名前空間 の [ネットワーク] タブで確認できます。

NSX 管理プロキシ サービスのトラブルシューティング

次のエラーが表示された場合は、環境に互換性がないことを意味します。環境が「前提条件」セクションに記載されているバージョンに準拠していることを確認してください。

```
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.
```

TKG サービス クラスタに対する Antrea-NSX Adapter アダプタの有効化

Antrea-NSX Adapter を有効にするには、このトピックを参照してください。これにより、Antrea CNI を使用する TKG サービス クラスタを NSX Manager と統合してネットワークの管理と監視を行うことができます。

Antrea-NSX Adapter の前提条件

次の前提条件を満たす必要があります。

- vSphere 8 U3 (8.0.3) 以降
- NSX 4.1 以降
- スーパーバイザー が NSX ネットワークで有効になっている
- TKG サービス 3.0 以降
- Tanzu Kubernetes リリース v1.28.x（vSphere 8.x 以降の場合）
- 管理ネットワークとワークロード ネットワークが分離されている、一般的な スーパーバイザー トポロジである場合は、NSX 管理プロキシ サービスのインストール

Antrea-NSX Adapter の要件

Antrea-NSX Adapter を使用すると、Antrea ベースの TKG サービス クラスタを NSX Manager と統合できます。アダプタを構成したら、NSX Manager を使用してクラスタのネットワーク動作を管理できます。Antrea-NSX 統合の機能の詳細については、『NSX 4.1 管理ガイド』の「Antrea CNI を含む Kubernetes クラスタとの統合」を参照してください。

NSX と統合する各 TKG サービス クラスタに対して Antrea-NSX Adapter を有効にする必要があります。つまり、アダプタとクラスタの比率が 1 対 1 になります。また、新しいクラスタのデプロイでのみ Antrea-NSX Adapter を使用できます。TKG サービス クラスタの作成前にアダプタを有効にする必要があります。また、作成するクラスタの名前をアダプタ リソース定義に含める必要があります。

TKGS クラスタの NSX Tier-0 または Tier 1 ゲートウェイが SNAT IP アドレスで構成されている場合、すべての Antrea-NSX 接続は単一の送信元 IP アドレスを共有します。NSX は、これを同じ IP アドレスからの複数の制御プレーン接続として解釈し、接続をドロップします。この場合は、NSX の UA ファイアウォール ルールを手動で変更する必要があります。詳細については、ナレッジベースの記事「https://knowledge.broadcom.com/external/article?articleNumber=317179」を参照してください。

Antrea-NSX Adapter の有効化

Antrea-NSX Adapter を有効にするには、次の手順を実行します。

1 kubectl を使用して、スーパーバイザー で認証します。

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-username USERNAME
```

2 TKG サービス クラスタをプロビジョニングするターゲット vSphere 名前空間 にコンテキストを切り替えます。

```
kubectl config use-context vsphere-namespace
```

3 `AntreaConfig.yaml` カスタム リソース定義を作成します。

```
#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
```

ここで、

- `antreaNSX.enable.true` 要素によって Antrea-NSX Adapter が有効になります。デフォルトでは、このフィールドは false です。
- `metadata.name` 要素には必須のサフィックス `-antrea-package` が含まれており、作成する TKG サービス クラスタ（アダプタを使用する TKG サービス クラスタ）の正確な名前がその前に配置されます。

4 AntreaConfig カスタム リソース定義を適用します。

```
kubectl apply -f AntreaConfig.yaml
```

5 TKG サービス クラスタをプロビジョニングします。

このアダプタは、v1alpha3 API または v1beta1 API のいずれかで使用できます。

Kubectl を使用して TKG クラスタをプロビジョニングするためのワークフローを参照してください。

6 NSX Manager にログインして、アダプタが機能していることを確認します。

詳細については、NSX 4.1 のドキュメントを参照してください。

Tanzu Kubernetes クラスタのデフォルト CNI の設定

Tanzu Kubernetes クラスタのデフォルトのコンテナ ネットワーク インターフェイス (CNI) は Antrea です。vSphere Client を使用すると、デフォルト CNI を変更できます。

デフォルトの CNI

vSphere IaaS control plane は、TKG クラスタについて、Antrea と Calico の 2 つの CNI オプションをサポートしています。システムで定義されているデフォルトの CNI は Antrea です。

デフォルトの CNI は、vSphere Client を使用して変更できます。デフォルトの CNI を設定するには、次の手順を実行します。

注意： デフォルトの CNI の変更は、グローバルに行われます。新しく設定されたデフォルトは、サービスによって作成されたすべての新規 TKG クラスタに適用されます。既存のクラスタは変更されません。

1 vSphere Client を使用して、vSphere IaaS control plane 環境にログインします。

2 [ワークロード管理]、[スーパーバイザー] の順に選択します。

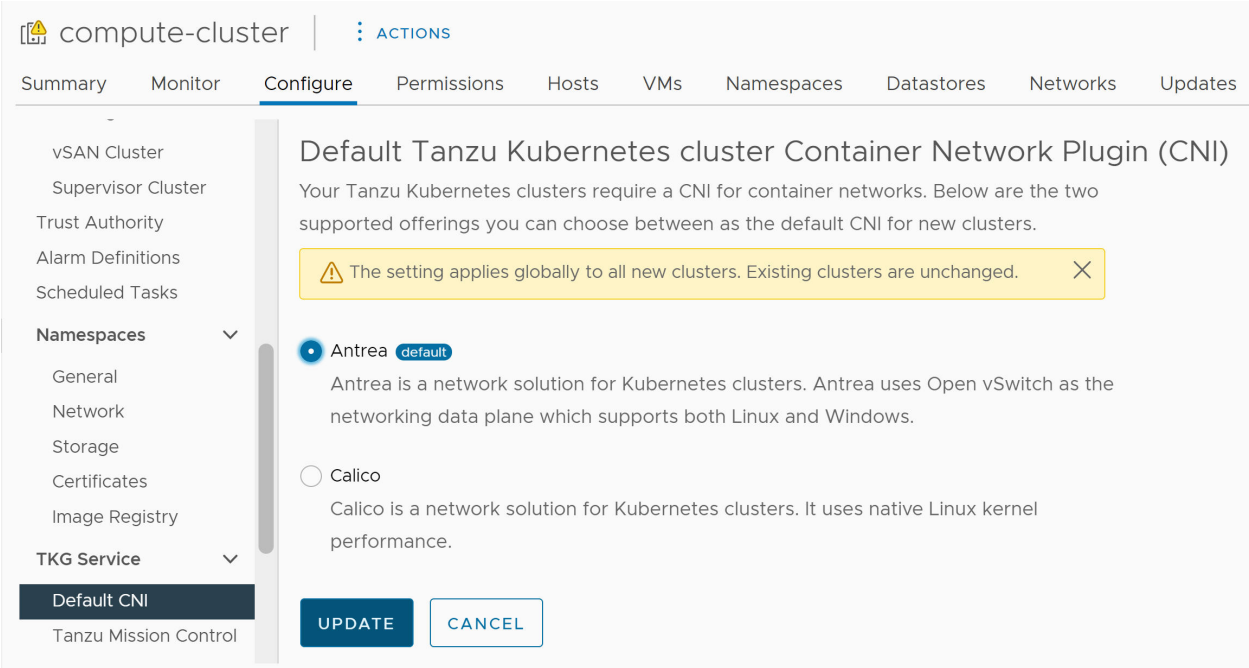
3 リストから スーパーバイザー インスタンスを選択します。

4 [構成] を選択し、[TKG サービス] - [デフォルトの CNI] の順に選択します。

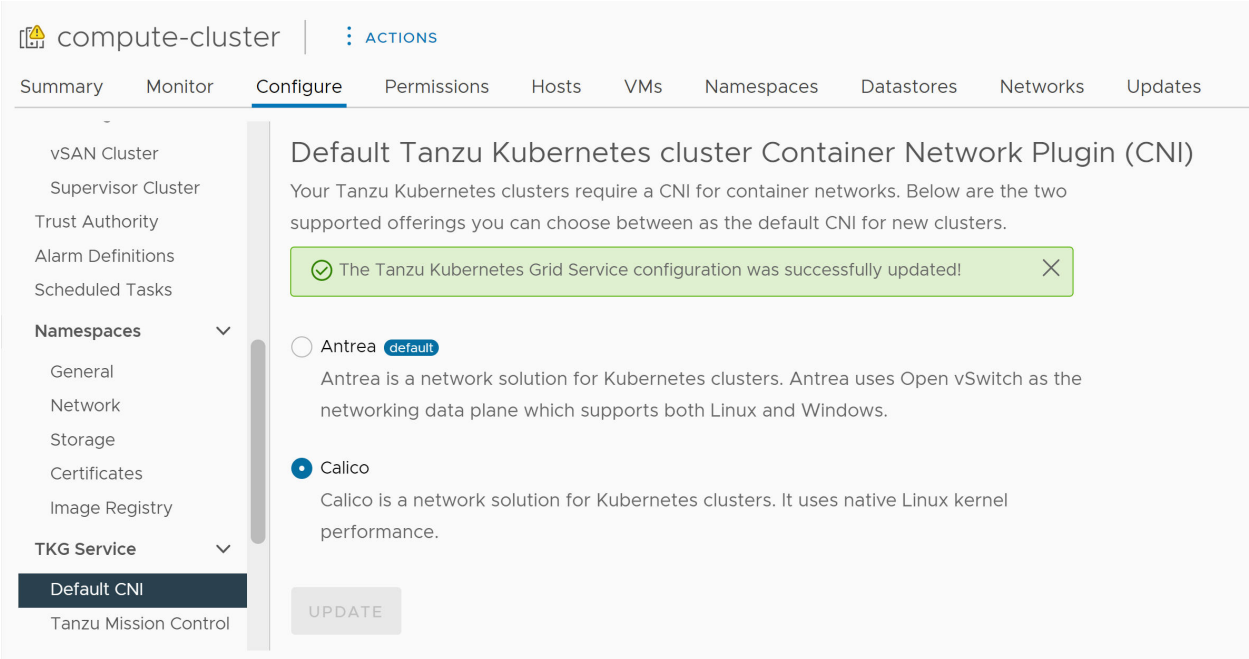
5 新規のクラスタに対してデフォルトの CNI を選択します。

6 [更新] をクリックします。

次の図は、デフォルトの CNI が選択されている様子を示しています。



次の図は、CNI の選択が Antrea から Calico に変更された様子を示しています。



TKG クラスタの TKG サービス構成のカスタマイズ

v1alpha3 API を使用してプロビジョニングされる TKG クラスタ向けに、コンテナ ネットワーク インターフェイス (CNI)、プロキシ サーバ、TLS 証明書などの TKG サービス構成をカスタマイズできます。

TkgServiceConfiguration のカスタマイズについて

TkgServiceConfiguration を編集して、Tanzu Kubernetes クラスタ向けにグローバル設定を構成できます。この構成により、デフォルト CNI の設定、グローバル プロキシ サーバの追加、1 つ以上の信頼された TLS 証明書の追加が可能になります。

注意： TkgServiceConfiguration のカスタマイズは、グローバルに行われます。`TkgServiceConfiguration` オブジェクトに対して行った変更は、そのサービスによってプロビジョニングされたすべての TKG クラスタに適用されます。ローリング アップデートが手動またはアップグレードによって開始された場合は、変更されたサービス仕様によってクラスタが更新されます。

注： TkgServiceConfiguration のカスタマイズは、v1alpha3 API を使用してプロビジョニングされた Tanzu Kubernetes クラスタに適用されます。v1beta1 API でプロビジョニングされたクラスタには適用されません。

TkgServiceConfiguration の仕様

`TkgServiceConfiguration` 仕様は、Tanzu Kubernetes Grid インスタンスを構成するためのフィールドを提供します。

重要： 有効なキー名は、英数字、ダッシュ（`key-name` など）、アンダースコア（`KEY_NAME` など）、ドット（`key.name` など）のみで構成する必要があります。キー名にスペースを使用することはできません。

```
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
```

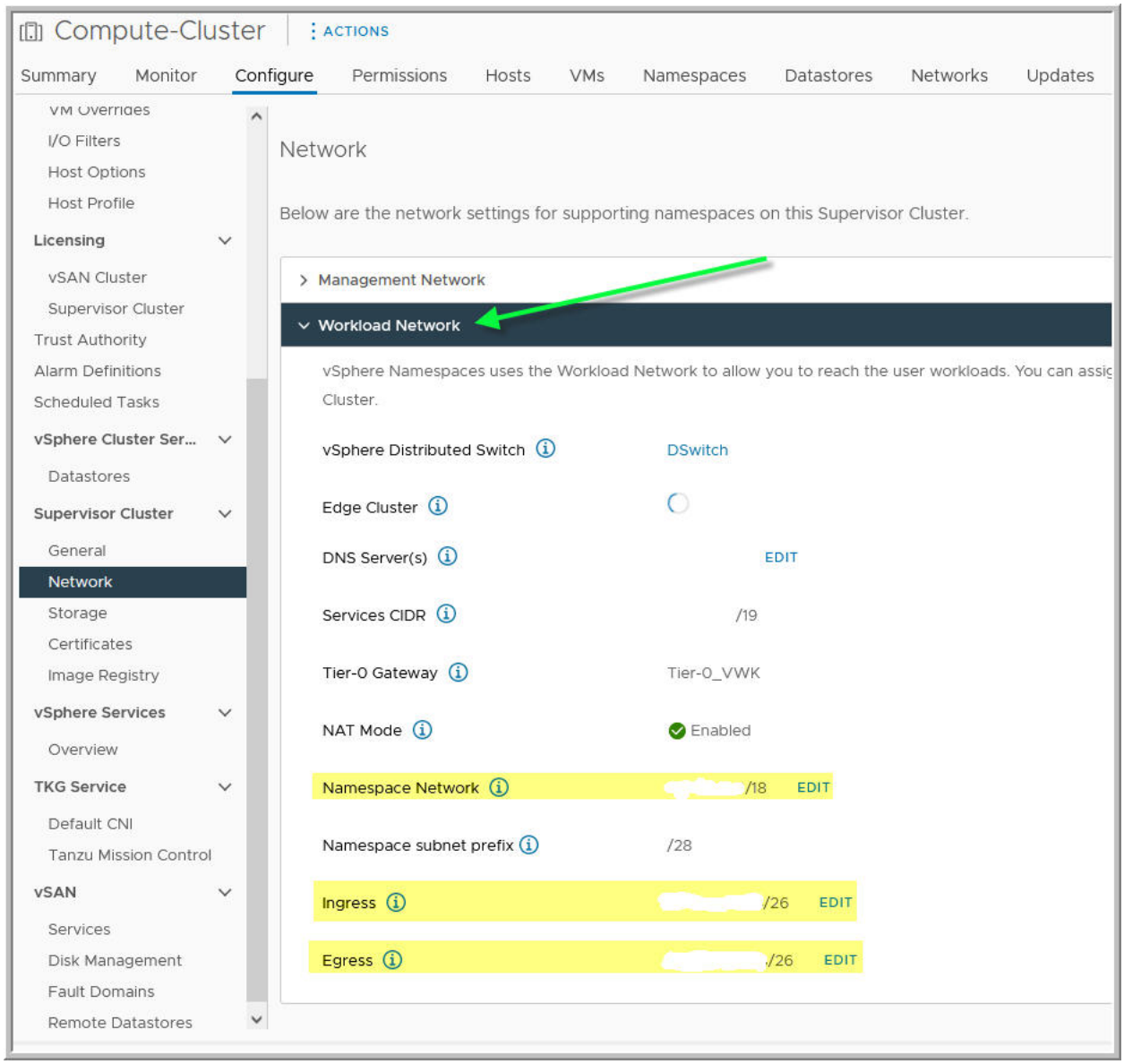
注釈付きの TkgServiceConfiguration 仕様

次の YAML は、`TkgServiceConfiguration` 仕様の各パラメータについて、構成可能なフィールドを示し、説明したものです。

```
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
```

図 17-1. 必須の noProxy 値



注： `TkgServiceConfiguration` でグローバル プロキシが構成されている場合、そのプロキシ情報は、クラスタの初期デプロイ後にクラスタ マニフェストに伝達されます。グローバル プロキシ構成は、クラスタの作成時にプロキシ構成フィールドがない場合にのみ、クラスタ マニフェストに追加されます。つまり、クラスタごとの構成が優先されるため、グローバル プロキシ構成は上書きされます。

TkgServiceConfiguration 仕様の例

次の YAML は、`TkgServiceConfiguration` 仕様の各パラメータについて、構成可能なフィールドを示し、説明したものです。

```
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
```

TkgServiceConfiguration の編集

TkgServiceConfiguration 仕様を編集するには、次の手順を参照してください。

1 Kubectl の編集を構成します。kubectl のテキスト エディタの構成を参照してください。

2 スーパーバイザー で認証します。

```
kubectl vsphere login --server=SVC-IP-ADDRESS --vsphere-username USERNAME
```

3 コンテキストをターゲット vSphere 名前空間 に切り替えます。

```
kubectl config use-context vSphere-Namespace
```

4 TkgServiceConfiguration 仕様を取得します。

```
kubectl get tkgserviceconfigurations
```

5 TkgServiceConfiguration 仕様を編集するためにロードします。

```
kubectl edit tkgserviceconfigurations tkg-service-configuration
```

`tkg-service-configuration` 仕様が、KUBE_EDITOR または EDITOR 環境変数によって定義されたデフォルトのテキスト エディタで開かれます。

6 要件に応じて TkgServiceConfiguration を編集します。

7 変更を適用するには、テキスト エディタでこのファイルを保存します。キャンセルするには、保存せずにエディタを閉じます。

テキスト エディタで変更を保存すると、kubectl によって `tkg-service-configuration` サービス仕様が更新されます。

8 TkgServiceConfiguration 仕様が更新されていることを確認します。

```
kubectl get tkgserviceconfigurations
```

既存のクラスタへのグローバル構成変更の伝達

`TkgServiceConfiguration` でグローバル レベルの設定を行っても、既存のクラスタにはその設定が自動的に伝達されない可能性があります。たとえば、`TkgServiceConfiguration` で `proxy` や `trust` の設定を変更しても、すでにプロビジョニングされているクラスタには影響しない可能性があります。

既存のクラスタにグローバルな変更を手動で伝達するには、Tanzu Kubernetes クラスタにパッチを適用して、`TkgServiceConfiguration` に対する変更をクラスタに継承させる必要があります。

例：

```
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}}}}"
```

TKG クラスタの NSX ネットワーク オブジェクト

このトピックでは、NSX ネットワークでの スーパーバイザー の使用時に TKG クラスタ用に作成されるネットワーク オブジェクトの一覧を示します。

TKG クラスタの NSX ネットワーク オブジェクト

各 TKG クラスタにはネットワーク リソース（仮想ネットワーク、仮想ネットワーク インターフェイス、仮想マシン サービス）が必要です。

vSphere IaaS control plane が有効な状態で スーパーバイザー のインスタンスがデプロイされると、NSX 組み込みロード バランサがシステムによって自動的にプロビジョニングされます。このロード バランサは、スーパーバイザー 制御プレーン用であり、Kubernetes API サーバへのアクセスを提供します。

TKG クラスタ用に LoadBalancer タイプの Kubernetes サービスを作成すると、そのサービスに NSX 組み込みロード バランサがプロビジョニングされます。

ネットワーク オブジェクト	ネットワーク リソース	説明
VirtualNetwork	Tier-1 ルーターと、リンクされたセグメント	クラスタのノード ネットワーク
VirtualNetworkInterface	セグメントの論理ポート	クラスタ ノードのノード ネットワーク インターフェイス
VirtualMachineService	該当なし	VirtualMachineService が作成され、k8s サービスに変換されます。
サービス	VirtualServer インスタンスおよび関連付けられているサーバ プール（メンバー プール）があるロード バランサ サーバ	TKG クラスタ API サーバにアクセスするために、ロード バランサ タイプの Kubernetes サービスが作成されます。

ネットワーク オブジェクト	ネットワーク リソース	説明
エンドポイント	エンドポイント メンバー（TKG クラスタ制御プレーン ノード）は、メンバー プールに含まれている必要があります。	すべての TKG クラスタ制御プレーン ノードを含むように、エンドポイントが作成されます。
スーパーバイザー上の VirtualMachineService	該当なし	VirtualMachineService がスーパーバイザーで作成され、スーパーバイザーで Kubernetes サービスに変換されます
スーパーバイザー上のロード バランサ サービス	TKG クラスタ ロード バランサ内の VirtualServer、および関連付けられているメンバー プール。	ロード バランサ サービスは、この LB タイプのサービスにアクセスするために、スーパーバイザー上に作成されます
スーパーバイザー上のエンドポイント	エンドポイント メンバー(TKG クラスタ ワーカー ノード）は、NSX のメンバー プールに含まれている必要があります。	すべての TKG クラスタ ワーカー ノードを含むように、エンドポイントが作成されます
TKG クラスタのロード バランサ サービス	該当なし	ユーザーによってデプロイされた TKG クラスタのロード バランサ サービスは、ステータスがロード バランサの IP で更新される必要があります

ノード ネットワーク

各 TKG クラスタには、次のネットワーク オブジェクトと関連する NSX リソースを作成しておく必要があります。

ネットワーク オブジェクト	NSX リソース	説明	IPAM
VirtualNetwork	Tier-1 ゲートウェイと、リンクされたセグメント	TKG クラスタのノード ネットワーク	SNAT IP アドレスが割り当てられている
VirtualNetworkInterface	リンクされたセグメントの論理ポート	TKG クラスタ ノードのノード ネットワーク インターフェイス	各ノードには IP アドレスが割り当てられます

制御プレーン ロード バランサ

ネットワーク オブジェクト	ネットワーク リソース	説明	IPAM
VirtualMachineService	該当なし	VirtualMachineService が作成され、Kubernetes サービスに変換されます。	ロード バランサの VIP が含まれる
サービス	VirtualServer インスタンスおよび関連付けられているサーバ プール（メンバー プール）があるロード バランサ サーバ	TKG クラスタ API サーバにアクセスするために、ロード バランサ タイプの Kubernetes サービスが作成されます。	外部 IP アドレスが割り当てられている
エンドポイント	エンドポイント メンバーは TKG クラスタ制御プレーン ノードであり、メンバー プールに含まれている必要があります。	すべての TKG クラスタ制御プレーン ノードを含むように、エンドポイントが作成されます。	該当なし

NSX ロード バランサ

作成された TKG クラスタごとに、システムは小規模な NSX ロード バランサの単一インスタンスを作成します。このロード バランサには、次の表に示すオブジェクトが含まれています。

オブジェクト番号	説明
1	ポート 8443 で Kubernetes 制御プレーン API にアクセスするための仮想サーバ (VS)。
1	3 つの Kubernetes 制御プレーン ノードを含むサーバ プール。
1	HTTP Ingress コントローラ用の VS。
1	HTTPS Ingress コントローラ用の VS。

NAT ルール

作成された TKG クラスタごとに、システムは Tier-0 論理ルーターで次の NSX NAT ルールを定義します。

オブジェクト番号	説明
1	変換された IP アドレスとしてフローティング IP アドレス プールの 1 つの IP アドレスを使用して、Kubernetes 名前空間ごとに作成される SNAT ルール。
1	（NAT トポロジのみ）変換された IP アドレスとしてフローティング IP アドレス プールの 1 つの IP アドレスを使用して、Kubernetes クラスタごとに作成される SNAT ルール。Kubernetes クラスタ サブネットは、/24 ネットマスクを使用してノードの IP ブロックから取得されます。

DFW ルール

作成された TKG クラスタごとに、システムは次の NSX Distributed Firewall ルールを定義します。

オブジェクト番号	説明
1	CoreDNS ポッドの論理ポートに適用される `kube-dns` 用の DFW ルール
1	Validator ポッドの論理ポートに適用される、名前空間内のバリデータ用の DFW ルール

TKG サービス クラスタのセキュリティの管理

18

このセクションでは、TKG サービス クラスタのセキュリティの管理について説明します。

次のトピックを参照してください。

- TKG サービス クラスタのセキュリティ
- TKR 1.25 以降の PSA の構成
- TKR 1.24 以前の PSP の構成
- TKG サービス クラスタへのデフォルトのポッド セキュリティ ポリシーの適用
- TKG サービス クラスタの TLS 証明書の管理
- NSX 証明書のローテーション

TKG サービス クラスタのセキュリティ

スーパーバイザー を使用する TKG サービスは vSphere のセキュリティ機能を活用して、デフォルトで安全なワークロード クラスタをプロビジョニングできます。

vSphere IaaS control plane は、vCenter Server および ESXi に組み込まれたセキュリティ機能を利用できる、vSphere に対するアドオン モジュールです。詳細については、vSphere Security ドキュメントを参照してください。

スーパーバイザー は、データベース (etcd) に保存されているすべてのシークレットを暗号化します。シークレットは、起動時に vCenter Server によって提供されるローカル暗号化キー ファイルを介して暗号化されます。復号キーは スーパーバイザー 制御プレーン ノードのメモリ (tempfs) に格納されるほかに、vCenter Server データベース内のディスクに暗号化形式で格納されます。各システムの root ユーザーは、復号キーをクリア テキストで入手できます。

同じ暗号化モデルは、各 TKG クラスタ制御プレーンにインストールされているデータベース (etcd) 内のデータに適用されます。すべての etcd 接続は、インストール時に生成され、アップグレード中にローテーションされる証明書を使用して認証されます。現在、証明書を手動でローテーションまたは更新することはできません。各ワークロード クラスタのデータベース内に保持されているシークレットは、クリア テキスト形式で保存されます。

TKG クラスタにインフラストラクチャ認証情報がありません。TKG クラスタ内に保存される認証情報では、TKG クラスタにテナントのある vSphere 名前空間 にのみアクセスが可能です。そのため、クラスタ オペレータまたはユーザーの権限のエスカレーションが行われることはありません。

TKG クラスタへのアクセスに使用される認証トークンは、スーパーバイザー または他の TKG クラスタへのアクセスに使用できないように範囲が設定されます。これにより、クラスタ オペレータ、またはクラスタを侵害する可能性があるユーザーは、TKG クラスタにログインするときに、root レベルのアクセス権を使用して vSphere 管理者のトークンをキャプチャできなくなります。

TKG クラスタはデフォルトでセキュリティ保護されています。Tanzu Kubernetes リリース v1.25 以降では、TKG クラスタのポッド セキュリティ アドミッション コントローラ (PSA) がデフォルトで有効になっています。v1.24 までの Tanzu Kubernetes リリース では、TKG クラスタに対して制限付きポッド セキュリティ ポリシー (PSP) を使用できます。開発者が特権ポッドまたはルート コンテナを実行する必要がある場合、クラスタ管理者は、最低でも、デフォルトの特権 PSP へのユーザー アクセスを許可するロールバインドを作成する必要があります。

TKR 1.25 以降の PSA の構成

Tanzu Kubernetes リリース v1.25 以降では、ポッド セキュリティ アドミッション (PSA) コントローラが有効になります。PSA により、名前空間ラベルを使用してポッド セキュリティを均一に適用できます。

TKR 1.25 以降で有効な PSA

ポッド セキュリティ アドミッション コントローラは Kubernetes のコントローラで、TKG クラスタで実行されているポッドにセキュリティ標準を適用することができます。デフォルトでは、Tanzu Kubernetes リリース v1.25 以降ではポッド セキュリティ アドミッション (PSA) コントローラが有効になります。廃止および削除されたポッド セキュリティ ポリシー (PSP) コントローラは PSA コントローラに置き換えられます。TKR 1.24 以前の PSP の構成も参照してください。

Tanzu Kubernetes リリース v1.25 は、警告するように構成された PSA を含む移行リリースです。Tanzu Kubernetes リリース v1.26 以降では、PSA が適用されます。TKG クラスタのアップグレードを想定して、ポッド ワークロードを PSP から PSA に移行する必要があります。ガイダンスについては、「Migrate from Pod Security Policy to the Built-In Pod Security Admission Controller」を参照してください。

PSA クラスタ全体の構成

vSphere 8 Update 3 以降では、v1beta1 API で使用可能な `podSecurityStandard` ClusterClass 変数を使用して、PSA クラスタ全体を構成できます。クラスタ v1beta1 API を参照してください。

PSA モード

PSA コントローラでは、`enforce`、`audit`、`warn` の 3 つのポッド セキュリティ モードがサポートされます。次の表に、各 PSA モードの一覧と説明を示します。

表 18-1. PSA モード

モード	説明
`enforce`	セキュリティ違反があると、ポッドが拒否されます。
`audit`	セキュリティ違反があると、監査ログに記録されたイベントに対する監査注釈の追加がトリガされますが、それ以外は許容されます。
`warn`	セキュリティ違反があると、ユーザーに対する警告がトリガされますが、それ以外は許容されます。

PSA 標準

PSA コントローラでは、セキュリティ範囲をカバーすることを目的とした 3 つのレベルのポッド セキュリティ標準が定義されています。ポッド セキュリティ標準は累積的で、許容から制限の範囲です。次の表に、各 PSA 標準の一覧と説明を示します。

表 18-2. PSA 標準

レベル	説明
`privileged`	最大限のレベルの権限を提供する無制限のコントロール。このセキュリティ標準では、既知の権限の昇格が許可されます。
`baseline`	最小限の制限があるコントロール。既知の権限の昇格は許可されません。このセキュリティ標準では、デフォルトの（最小限の指定がされた）ポッド構成が許可されます。
`restricted`	ポッドのセキュリティ強化のベスト プラクティスに従って、コントロールが厳しく制限されています。

PSA 名前空間ラベル

PSA コントローラは、ポッド セキュリティを Kubernetes 名前空間のレベルで適用します。名前空間ラベルを使用して、特定の名前空間のポッドに使用する PSA モードとレベルを定義します。

Kubernetes には、名前空間に使用する標準の定義に使用できる一連のラベルが用意されています。適用するラベルは、PSA 違反が検出された場合に Kubernetes 制御プレーンが実行するアクションを定義します。特定の Kubernetes 名前空間では、任意またはすべてのモードを構成することも、異なるモードに異なるレベルを設定することもできます。

PSA 名前空間ラベルの構文は次のとおりです。

```
# MODE must be one of `enforce`, `audit`, or `warn`.
# LEVEL must be one of `privileged`, `baseline`, or `restricted`.
pod-security.kubernetes.io/<MODE>=<LEVEL>
```

また、モードごとのバージョン ラベルを適用することにより、Kubernetes バージョンにセキュリティ標準を結び付けることもできます。詳細については、「Enforce Pod Security Standards with Namespace Labels」を参照してください。

TKG クラスタのデフォルトの PSA

デフォルトでは、Tanzu Kubernetes リリース v1.25 を使用してプロビジョニングされた TKG クラスタでは、システム以外の名前空間の PSA モードである `warn` と `audit` が `restricted` に設定されます。これは強制なしの設定です。ポッドがセキュリティに違反している場合、PSA コントローラは警告と監査通知を生成しますが、ポッドは拒否されません。

デフォルトでは、Tanzu Kubernetes リリース v1.26 以降を使用してプロビジョニングされた TKG クラスタでは、システム以外の名前空間の PSA モードである `enforce` が `restricted` に設定されます。セキュリティに違反しているポッドは拒否されます。制限の少ないコントロールでポッドを実行するには、名前空間で PSA を構成する必要があります。

重要： kube-system、tkg-system、および vmware-system-cloud-provider 名前空間で実行されている一部のシステム ポッドには、昇格した権限が必要です。これらの名前空間はポッド セキュリティから除外されます。また、システム名前空間のポッド セキュリティは変更できません。

次の表に、TKG クラスタのデフォルトの PSA 構成を示します。

表 18-3. TKG クラスタのデフォルトの PSA

TKr バージョン	デフォルトの PSA
TKr v1.25	モード：warn	レベル：restricted モード：audit	レベル：restricted モード：enforce	レベル：未設定
TKr v1.26 以降	モード：enforce	レベル：restricted

名前空間ラベルを使用した PSA の構成

Tanzu Kubernetes リリース v1.25 の場合は、次の例のコマンドを使用して、PSA 警告と監査通知が生成されないように、特定の名前空間のセキュリティ レベルを変更します。

```
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/audit=privileged
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/warn=privileged
```

Tanzu Kubernetes リリース v1.26 以降の場合は、次の例のコマンドを使用して、PSA 標準を `restricted` から `baseline` にダウングレードします。

```
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/enforce=baseline
```

たとえば、デフォルトの名前空間に `baseline` 標準を適用するには、次のコマンドを実行します。

```
kubectl label --overwrite ns default pod-security.kubernetes.io/enforce=baseline
```

Tanzu Kubernetes リリース v1.26 以降の場合は、次の例のコマンドを使用して、PSA 標準を `restricted` から `privileged` にダウングレードします。

```
kubectl label --overwrite ns NAMESPACE pod-security.kubernetes.io/enforce=privileged
```

たとえば、デフォルトの名前空間に `privileged` 標準を適用するには、次のコマンドを実行します。

```
kubectl label --overwrite ns default pod-security.kubernetes.io/enforce=privileged
```

Tanzu Kubernetes リリース v1.26 以降では、次の例のコマンドを使用して、システム以外のすべての名前空間で PSA を緩和します。

```
kubectl label --overwrite ns --all pod-security.kubernetes.io/enforce=privileged
```

```
kubectl label --overwrite ns --all pod-security.kubernetes.io/warn=restricted
```

個々のポッドのセキュリティ コンテキストの構成

PSA に違反するポッドを実行すると、そのことを示すエラー メッセージが返されます。次に例を示します。

```
{"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":[]}
```

名前空間全体に PSA を設定する代わりに、個々のポッドのセキュリティ コンテキストを構成できます。個々のポッドを実行できるようにするには、ポッド仕様で次のようにセキュリティ コンテキストを設定します。

```
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
```

TKR 1.24 以前の PSP の構成

スーパーバイザー 上の TKG は、TKR v1.24 以前を使用する Tanzu Kubernetes クラスタではデフォルトで有効になっているポッド セキュリティ ポリシー アドミッション コントローラを使用して、ポッド セキュリティをサポートします。

前提条件：TKR 1.24 以前

このトピックの内容は、TKR v1.24 以前でプロビジョニングされた スーパーバイザー 上の TKG クラスタに適用されます。これらの Tanzu Kubernetes リリース では、デフォルトでポッド セキュリティ ポリシー アドミッション コントローラが有効になっています。

ポッド セキュリティ ポリシー アドミッション コントローラの後継は、ポッド セキュリティ アドミッション コントローラです。TKR v1.25 以降でプロビジョニングされた スーパーバイザー 上の TKG クラスタでは、ポッド セキュリティ アドミッション コントローラが有効になります。TKR 1.25 以降の PSA の構成を参照してください。

ポッド セキュリティ ポリシー アドミッション コントローラ

Kubernetes のポッド セキュリティ ポリシー (PSP) は、ポッドのセキュリティを制御するクラスタレベルのリソースです。PSP を使用すると、デプロイできるポッドのタイプと、それらをデプロイできるアカウントのタイプを制御できます。

PodSecurityPolicy リソースは、デプロイのためにポッドが満たす必要のある必要な一連の条件を定義しています。条件が満たされていない場合は、ポッドをデプロイできません。1 つの PodSecurityPolicy で 1 つのポッド全体を検証する必要があります。ポッドのルールを複数のポリシーに分けて定義することはできません。詳細については、Kubernetes のドキュメントでポッド セキュリティ ポリシーと RBAC について参照してください。

Kubernetes では、さまざまな方法でポッド セキュリティ ポリシーを使用できます。一般的なアプローチは、ロールベースのアクセス コントロール (RBAC) オブジェクトを使用するものです。ClusterRole および ClusterRoleBinding は、クラスタ全体に適用されます。Role および RoleBinding は、特定の名前空間に適用されます。RoleBinding を使用すると、ポッドはこのバインドと同じ名前空間内でのみ実行されます。詳細については、Kubernetes のドキュメントで RBAC を参照してください。

Kubernetes ポッドの作成には、直接的な方法と間接的な方法の 2 つがあります。ユーザー アカウントを使用してポッドの仕様をデプロイすることにより、ポッドを直接作成します。Deployment や DaemonSet などの上位レベルのリソースを定義することにより、ポッドを間接的に作成します。この場合、基盤となるポッドがサービス アカウントによって作成されます。詳細については、Kubernetes のドキュメントでサービス アカウントを参照してください。

PSP を効果的に使用するには、両方（直接および間接的）のポッド作成ワークフローを考慮する必要があります。ユーザーがポッドを直接作成すると、そのユーザー アカウントにバインドされた PSP によって操作が制御されます。ユーザーがサービス アカウントを使用してポッドを作成する場合、PSP は、ポッドの作成に使用するサービス アカウントにバインドする必要があります。ポッド仕様でサービス アカウントが指定されていない場合は、名前空間のデフォルトのサービス アカウントが使用されます。

TKG クラスタのデフォルトの PodSecurityPolicy

次の表に、TKG クラスタについて権限のある、および制限されたデフォルトのポッド セキュリティ ポリシーと、各ポリシーに関連付けられているデフォルトの ClusterRole の一覧と説明を示します。

表 18-4. デフォルトの PodSecurityPolicy および関連付けられている ClusterRole

デフォルトの PSP	権限	説明	関連付けられているデフォルトの ClusterRole
`vmware-system-privileged`	任意のユーザーとして実行	許可的な PSP。PSP アドミッション コントローラが有効になっていないクラスタの実行と同等です。	`psp:vmware-system-privileged` がこの PSP を使用可能
`vmware-system-restricted`	非 root として実行	制限的な PSP。ポッド コンテナへの特権アクセスを許可せず、root へのエスカレーションをブロックし、いくつかのセキュリティ メカニズムを使用する必要があります。	`psp:vmware-system-restricted` がこの PSP を使用可能

TKG クラスタの Role と ClusterRole のバインド

スーパーバイザー 上の TKG には、TKG クラスタのデフォルトの RoleBinding と ClusterRoleBinding はありません。使用例は、このドキュメントに記載されています。TKG サービス クラスタへのデフォルトのポッド セキュリティ ポリシーの適用を参照してください。

vSphere 名前空間 に対する編集権限が付与されている vCenter Single Sign-On ユーザーは、その名前空間でデプロイされるすべての Tanzu Kubernetes クラスタで cluster-admin ロールに割り当てられます。認証されたクラスタ管理者は、暗黙的に `vmware-system-privileged` PSP を使用できます。厳密には ClusterRoleBinding とは異なりますが、実質的には同じです。

クラスタ管理者は、ユーザーがクラスタにデプロイできるポッドのタイプを許可または制限するために、何らかのバインドを定義する必要があります。RoleBinding を使用すると、バインドと同じ名前空間でのみポッドの実行が許可されます。これをシステム グループと組み合わせると、名前空間で実行されているすべてのポッドへのアクセスを許可することができます。クラスタに対して認証を行う管理者以外のユーザーは、`authenticated` ロールに割り当てられ、デフォルトの PSP にバインドできます。

ポッド セキュリティ ポリシーを必要とする TKG クラスタには、次の動作が適用されます。

- クラスタ管理者はそれぞれのユーザー アカウントを使用して、任意の名前空間に特権ポッドを直接作成できます。
- クラスタ管理者は、kube-system 名前空間に Deployments、StatefulSets、および DaemonSet（それぞれが特権ポッドを作成）を作成できます。別の Kubernetes 名前空間を使用する場合は、その名前空間の RoleBinding または ClusterRoleBinding を作成します。
- クラスタ管理者は、独自の PSP を（2 つのデフォルトの PSP に加えて）作成して任意のユーザーにバインドすることができます。独自の PSP を定義する場合は、Kubernetes のドキュメントでポリシーの順序について参照してください。
- クラスタ管理者が PSP を認証されたユーザーにバインドするまで、認証されたユーザーは、特権の有無にかかわらずポッドを作成することはできません。

バインドの例については、TKG サービス クラスタへのデフォルトのポッド セキュリティ ポリシーの適用を参照してください。

TKG サービス クラスタへのデフォルトのポッド セキュリティ ポリシーの適用

TKR 1.24 以前を使用している TKG サービス クラスタには、権限および制限のあるワークロード デプロイ用の、バインド可能なポッド セキュリティ ポリシーがデフォルトで含まれています。

ロールと ClusterRole バインドを使用したデフォルトのポッド セキュリティ ポリシーの適用

vSphere IaaS control plane には TKR 1.24 以前の PSP の構成が用意されており、TKR 1.24 以前を実行している スーパーバイザー 上の TKG クラスタに適用できます。これを行うには、TKR 1.24 以前の PSP の構成を参照する RoleBinding および ClusterRoleBinding オブジェクトを作成します。

注： 独自のポッド セキュリティ ポリシーを作成するには、Kubernetes のドキュメントを参照してください。

RoleBinding は、特定の名前空間内で権限を付与します。ClusterRoleBinding は、クラスタ全体に権限を付与します。RoleBindings または ClusterRoleBinding を使用するかどうかは、使用方法によって異なります。たとえば、ClusterRoleBinding を使用して、`system:serviceaccounts:<namespace>` を使用するようにサブジェクトを構成する場合は、名前空間を作成する前に PSP にバインドできます。詳細については、Kubernetes ドキュメントの RoleBinding および ClusterRoleBinding を参照してください。

以下のセクションでは、TKR 1.24 以前の PSP の構成の使用を許可する RoleBinding および ClusterRoleBinding オブジェクトを作成するための YAML および CLI コマンドについて説明します。

例 1：権限のあるワークロードのセットを実行するための ClusterRoleBinding

次の kubectl コマンドにより、ClusterRoleBinding が作成されます。これは、デフォルトの PSP `vmware-system-privileged` を使用して権限のあるワークロードのセットを実行する認証済みユーザーに、アクセス権を付与します。

注意: 例 1 を宣言または命令によって適用すると、クラスタ全体で権限を付与されたワークロードのデプロイが可能になります。実際、例 1 では、ネイティブのセキュリティ制御が無効になるため、注意を払い、影響を十分に認識して使用する必要があります。セキュリティを強化するには、例 2、3、および 4 を検討してください。

```
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
```

YAML を適用する代わりに、次の `kubectl` コマンドを実行することもできます。

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding --
clusterrole=psp:vmware-system-privileged --group=system:authenticated
```

例 2：権限のあるワークロードのセットを実行するための RoleBinding

次の kubectl コマンドは、デフォルトの PSP `vmware-system-privileged` を使用して、権限のあるワークロードのセットを実行するためにデフォルトの名前空間内のすべてのサービス アカウントへのアクセスを許可する RoleBinding を作成します。

```
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
```

YAML を適用する代わりに、次の kubectl コマンドを実行することもできます。

```
kubectl create rolebinding rolebinding-default-privileged-sa-ns_default --namespace=default --
clusterrole=psp:vmware-system-privileged --group=system:serviceaccounts
```

例 3：制限されたワークロードのセットを実行するための ClusterRoleBinding

次の YAML は、デフォルトの PSP `vmware-system-restricted` を使用して、制限されたワークロードのセットを実行するためにクラスタ全体へのアクセスを認証ユーザーに許可する ClusterRoleBinding を作成します。

```
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
```

YAML を適用する代わりに、次の kubectl コマンドを実行することもできます。

```
kubectl create clusterrolebinding psp:authenticated --clusterrole=psp:vmware-system-
restricted --group=system:authenticated
```

例 4：制限されたワークロードのセットを実行するための RoleBinding

次の YAML は、デフォルトの PSP `vmware-system-restricted` を使用して、制限されたワークロードのセットを実行するために特定の名前空間内のすべてのサービス アカウントへのアクセスを許可する RoleBinding を作成します。

```
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
```

YAML を適用する代わりに、次の kubectl コマンドを実行することもできます。

注： 「some-namespace」をターゲットの名前空間に変更します。

```
kubectl create rolebinding psp:serviceaccounts --clusterrole=psp:vmware-system-restricted --
group=system:serviceaccounts -n some-namespace
```

ポッド セキュリティ ポリシーのロールの例

ワークロードをデプロイするために独自のポッド セキュリティ ポリシー (PSP) を定義する場合、この例を参照して、カスタム PSP を参照するロールまたは ClusterRole を作成します。

例は、PodSecurityPolicy にバインドされたロールを示しています。ロールの定義で `use` を verb として使用して、定義するカスタム PSP リソースに `example-role` を付与します。または、デフォルトの PSP の 1 つを使用します。次に、バインドを作成します。

```
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
```

TKG サービス クラスタの TLS 証明書の管理

vSphere IaaS control plane は、トランスポート レイヤー セキュリティ (TLS) 暗号化を使用してコンポーネント間の通信を保護します。スーパーバイザー 上の TKG には、この暗号化インフラストラクチャをサポートするいくつかの TLS 証明書が含まれています。スーパーバイザー 証明書のローテーションは手動で行います。TKG 証明書のローテーションは自動化されていますが、必要に応じて手動で行うこともできます。

TKG サービス クラスタの TLS 証明書について

vSphere IaaS control plane は、TLS 証明書を使用して、次のコンポーネント間の通信を保護します。

- vCenter Server
- スーパーバイザー 制御プレーン ノード

- vSphere ポッド のワーカー ノードとして機能している ESXi ホスト
- TKG クラスタ ノード（制御プレーンとワーカーの両方）

vSphere IaaS control plane は、システム ユーザーを認証するために次の信頼ドメインで動作します。

信頼ドメイン	説明
vCenter Server の信頼ドメイン	この信頼ドメインの TLS 証明書のデフォルトの署名者は、vCenter Server に組み込まれている VMware Certificate Authority (VMCA) です。
Kubernetes の信頼ドメイン	この信頼ドメインの TLS 証明書のデフォルトの署名者は、Kubernetes Certificate Authority (CA) です。

vSphere IaaS control plane 環境で使用される各 TLS 証明書のさらに詳しい情報と完全なリストについては、ナレッジベースの記事「vSphere with Tanzu Certificate Guide」を参照してください。

TLS 証明書のローテーション

TLS 証明書をローテーションする手順は、証明書が スーパーバイザー と TKG サービス クラスタのどちらで使用されるかによって異なります。

スーパーバイザー証明書のローテーション

スーパーバイザー の TLS 証明書は、VMCA 証明書から取得されます。スーパーバイザー 証明書の詳細については、ナレッジベースの記事「vSphere with Tanzu Certificate Guide」を参照してください。

スーパーバイザー の証明書のローテーションは手動で行います。WCP Cert Manager ツールを使用して スーパーバイザー 証明書を置き換える方法については、ナレッジベースの記事「Replace vSphere with Tanzu Supervisor Certificates」を参照してください。

TKG 2.0 クラスタ証明書のローテーション

通常、TKG クラスタの TLS 証明書を手動でローテーションする必要はありません。TKG クラスタを更新すると、ローリング アップデート プロセスによって、TLS 証明書が自動的にローテーションされるためです。

TKG クラスタの TLS 証明書の有効期限が切れていない場合に、これらの証明書を手動でローテーションする必要がある場合は、次のセクションの手順を実行します。

TKG サービス クラスタの TLS 証明書の手動ローテーション

これらの手順を行うには、TKG クラスタ管理に関する高度な知識と経験が必要です。また、これらの手順では、TLS 証明書が期限切れになっていないことが前提となります。証明書が期限切れになっている場合は、次の手順を実行しないでください。

1 これらの手順を実行するには、スーパーバイザー ノードのいずれかに SSH 接続します。Kubernetes 管理者およびシステム ユーザーとしての TKG サービス クラスタへの接続を参照してください。

2 TKG クラスタ名を取得します。

```
export CLUSTER_NAMESPACE="tkg-cluster-ns"

kubectl get clusters -n $CLUSTER_NAMESPACE
NAME                    PHASE         AGE   VERSION
tkg-cluster             Provisioned   43h
```

3 TKG クラスタの kubeconfig を取得します。

```
export CLUSTER_NAME="tkg-cluster"

kubectl get secrets -n $CLUSTER_NAMESPACE $CLUSTER_NAME-kubeconfig -o
jsonpath='{.data.value}' | base64 -d > $CLUSTER_NAME-kubeconfig
```

4 TKG クラスタの SSH キーを取得します。

```
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 証明書ローテーションを行う前に環境を確認します。

```
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;
```

前のコマンドの結果の例：

```
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 TKG 2.0 クラスタの TLS 証明書をローテーションします。

次の手順に進む前に、コンテキストを スーパーバイザー に戻します。

```
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
```

マシン ロールアウトの開始：

```
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
```

マシン ロールアウトの完了：

```
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 TKG 2.0 クラスタの手動による証明書のローテーションを確認します。

次のコマンドを実行して、証明書のローテーションを確認します。

```
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;
```

更新された有効期限を示す結果の例。

```
######
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 Kubelet 証明書を確認します。

Kubelet 構成のパラメータ `rotateCertificates` がデフォルトの構成である `true` に設定されている場合、Kubelet 証明書をローテーションする必要はありません。

この構成は、次のコマンドを使用して確認できます。

```
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;
```

結果の例：

```
######
tkg-cluster-workers-v8575-59c6645b4-wvnlz
rotateCertificates: true
```

NSX 証明書のローテーション

スーパーバイザー は、スーパーバイザー と NSX 間の通信に TLS を使用します。NSX ネットワーク スタックを使用して スーパーバイザー をデプロイした場合、さまざまな NSX 証明書でローテーションが必要になる可能性があります。

スーパーバイザー で使用される NSX 証明書

NSX を使用する WCP では、NSX との統合に 2 つの証明書が使用されます。

- NSX ロード バランサの証明書とキー
- NSX Manager の証明書とキー

これらの証明書の詳細については、『NSX 管理ガイド』の NSX フェデレーションの証明書を参照してください。

注： このトピックの情報は、NSX v3.2 に基づいています。

NSX ロード バランサの証明書とキーのローテーション

[スーパーバイザー] - [証明書] - [NSX ロード バランサ] 画面で、NSX ロード バランサの TLS 証明書とキーをローテーションできます。

- [アクション] - [CSR の生成] の順に選択して、証明書を生成します。
- [アクション] - [証明書の置き換え] の順に選択して、証明書とキーを更新します。

各 NSX Manager ノードの自己署名証明書とキーの生成

スーパーバイザー は、エンタープライズ管理者アカウントを使用して NSX Manager API にアクセスします。NSX Manager 証明書が期限切れになると、スーパーバイザー は NSX にアクセスできなくなります。

NSX Manager 証明書が期限切れになった場合は、スーパーバイザー ログを確認してください。

```
tail -f /var/log/vmware/wcp/wcpsvc.log
```

次のようなエラーが記録されることがあります。

```
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
```

この問題を解決するには、各 NSX Manager ノードの証明書とキーを更新します。VIP アドレスを持つ 3 ノード NSX 管理クラスタを使用している場合、スーパーバイザー では VIP アドレスが使用されないことに注意してください。つまり、各 NSX Manager ノードで各証明書をローテーションする必要があります。VIP 証明書のみを置き換えて証明書をローテーションすることはできません。

1 NSX Manager ノードの証明書をローテーションするには、指定された証明書署名要求を作成し、以下の内容をポピュレートします。

ここで、

- `NSX-MGR-IP-ADDRESS` は NSX Manager の IP アドレス
- `NSX-MGR-FQDN` は、NSX Manager の FQDN または IP アドレス

`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
```

例：

```
[ 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 OpenSSL を使用して SSL 証明書とプライベート キーを生成します。

```
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 コマンドを実行した後に、次の出力が表示されることを確認します。

```
Generating a 2048 bit RSA private key
..............+++
..............+++
writing new private key to 'nsx-mgr-01.key'
```

4 署名要求を実行すると、最初の署名要求、証明書とプライベート キーの 3 つのファイルが生成されます。

5 次のコマンドを実行して、証明書とプライベート キーを確認します。

```
openssl x509 -in nsx-mgr-01.crt -text -noout
```

6 マルチノード NSX 管理クラスタを使用している場合は、NSX Manager ノードごとにこのプロセスを繰り返します。証明書署名要求に含まれる IP アドレスと FQDN および出力ファイル名を適宜変更します。

NSX-T 管理コンソールへの SSL 証明書とプライベート キーのインポート

次の手順を実行して、各 NSX Manager ノードの証明書とプライベート キーを NSX にインポートします。`nsx.crt` および `nsx.key` ファイルをローカルに保存する場合は、ファイルを NSX にアップロードするか、内容をコピーして貼り付けます。

1 NSX 管理コンソールにログインし、[システム] - [証明書] 画面に移動します。

2 [インポート] - [証明書のインポート] をクリックします。

注： 自己署名証明書を生成した後で、[CA 証明書のインポート] ではなく [証明書のインポート] を選択してください。

3 証明書とキー ペア（`nsx-mgr-01-cert-and-key` など）にわかりやすい [名前] を入力します。

4 証明書ファイルを参照して選択するか、ヘッダーやフッターなどの内容をコピーして貼り付けます。

例：

```
-----BEGIN CERTIFICATE-----
MIID+zCCAuOgAwIBAgIUCfXaWxNwXvrEFQbt+Dvvp9C/UkIwDQYJKoZIhvcNAQEL
BQAwVTELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExCzAJBgNVBAcM
...
FGlnyT4vxpa2TxvXNTCuXPV9z0VtVBF2QpUJluGH7W1i2wUnApCCXhItcBkfve0f
pCi9YoRoUT8fuMBYo7sL
-----END CERTIFICATE-----
```

5 キーを参照して選択するか、ヘッダーやフッターなどの内容をコピーして貼り付けます。

例：

```
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQC5GNlUSYHa1p+E
XuOGAsIgiFxUnerRYNm2ARMqRb/xKK6R4xgZhBmpmikpE90vQibvouHqnL13owq7
...
OzbnwMCUI2TeY1iJNx3HNKUrdLvrr8CMh7Looe0L4/2j7ygew2x2C5m272SCJYs/
ly+bOXEYaH4/ORHbvvr0jQ==
-----END PRIVATE KEY-----
```

6 [サービス証明書] オプションに対して [いいえ] を選択します。

7 証明書とキー ペア（`Cert and Private Key for NSX Manager Node 1` など）に説明を入力します。

8 [インポート] をクリックします。

9 NSX Manager の証明書とキーのペアごとに、このプロセスを繰り返します。

NSX API を使用した NSX Manager 証明書の登録

証明書とキーを NSX Manager にアップロードした後に、NSX API を使用して登録します。『NSX 管理ガイド』の証明書のインポートと置き換えも参照してください。

1 NSX Manager で、[システム] - [証明書] の順に選択します。

2 [ID] 列で、登録する証明書の ID を選択し、ポップアップ ウィンドウから証明書 ID をコピーします。

3 次の API 呼び出しを実行して、証明書を一覧表示します。更新する証明書ごとに証明書ノード ID を取得します。

```
GET https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates
```

4 次の API 呼び出しを実行して、証明書を検証します。

```
GET https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates/<cert-id>?
action=validate
```

例：

```
https://10.19.92.133/api/v1/trust-management/certificates/070bae44-7548-45ff-
a884-578f079eb6d4?action=validate
```

5 次の API 呼び出しを実行して、NSX Manager ノードの証明書を置き換えます。

```
POST https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates/<cert-id>?
action=apply_certificate&service_type=API&node_id=<node-id>
```

例：

```
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 マルチノード NSX 管理クラスタを使用している場合は、NSX Manager ノードごとに証明書置き換えプロセスを繰り返します。

7 完了したら、置き換えた期限切れの各証明書を削除します。この操作は、NSX Manager インターフェイスまたは NSX API を使用して実行できます。

例：

```
https://NSX-MGR-IP-or-FQDN/api/v1/trust-management/certificates/<cert-id>
```

TMC と TKG サービス クラスタの統合

19

このセクションには、Tanzu Mission Control と TKG サービス クラスタの統合に関するトピックが記載されています。

次のトピックを参照してください。

- ホスト型 Tanzu Mission Control の スーパーバイザー への登録
- Tanzu Mission Control Self-Managed の スーパーバイザー への登録

ホスト型 Tanzu Mission Control の スーパーバイザー への登録

スーパーバイザー 上の Tanzu Kubernetes Grid を Tanzu Mission Control と統合できます。それにより、TMC Web インターフェイスを使用して TKG サービス クラスタを管理できます。

TKG サービス クラスタと Tanzu Mission Control の統合

TKG サービス クラスタを Tanzu Mission Control および Tanzu Application Platform と統合するための一般的なワークフローは次のとおりです。

手順	操作
1	TKG ワークロード クラスタをプロビジョニングします 7 章 TKG サービス クラスタのプロビジョニングを参照してください
2	スーパーバイザー を TMC に登録します スーパーバイザー を Tanzu Mission Control に登録しますを参照してください
3	TKG ワークロード クラスタを TMC に接続します 詳細については、Tanzu Kubernetes クラスタのライフサイクルの管理を参照してください
4	Tanzu Application Platform をインストールします Tanzu Application Platform のドキュメントを参照してください

スーパーバイザー を Tanzu Mission Control に登録します

スーパーバイザー には、Tanzu Mission Control 専用の vSphere 名前空間 が付属しています。スーパーバイザー を Tanzu Mission Control に登録するには、TMC Agent を vSphere 名前空間 にインストールし、TMC を使用して登録プロセスを完了します。

TMC エージェントを スーパーバイザー にインストールするには、次の手順を実行します。この手順では、vSphere Client と TMC Web インターフェイスを同時に使用する必要があることに注意してください。

1 kubectl 向けの vSphere プラグイン を使用して、スーパーバイザー での認証を行います。

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

2 次のように、コンテキストを スーパーバイザー に切り替えます。

```
kubectl config use-context 10.199.95.59
```

3 次のコマンドを実行して、名前空間を一覧表示します。

```
kubectl get ns
```

4 TMC の vSphere 名前空間 には、`svc-tmc-cXX` という名前が付けられます（XX は数字）。これが存在し、有効であることを確認します。

5 Tanzu Mission Control Web インターフェイスにログインします。

6 スーパーバイザー を Tanzu Mission Control に登録します。管理クラスタの Tanzu Mission Control への登録を参照してください。

7 Tanzu Mission Control Web インターフェイスを使用して、[管理] - [管理クラスタ] 画面で選択できる スーパーバイザー の登録 URL をコピーします。

8 必要に応じて、Tanzu Mission Control が必要とする スーパーバイザー のファイアウォール ポート（通常は 443）を開きます。クラスタ エージェント拡張機能によって確立される送信接続を参照してください。

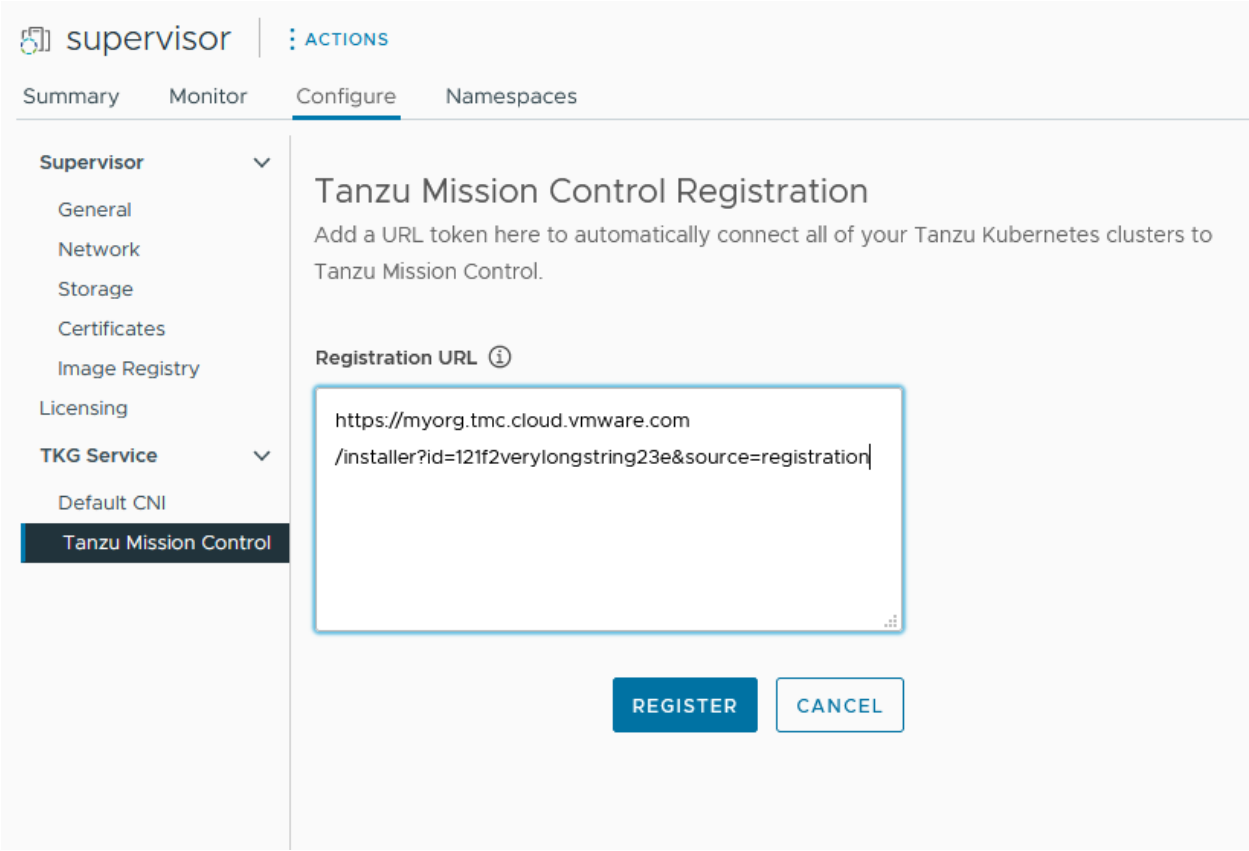
9 vSphere Client を使用して、vSphere IaaS control plane 環境にログインします。

10 [ワークロード管理] を選択してから、ターゲットの スーパーバイザー を選択します。

11 [構成] を選択し、[TKG サービス] - [Tanzu Mission Control] の順に選択します。

12 [登録 URL] フィールドに登録 URL を貼り付けます。

13 [登録] をクリックします。



スーパーバイザー が TMC に登録されたら、TMC Web インターフェイスを使用して TKG クラスタをプロビジョニングおよび管理します。手順については、Tanzu Mission Control のドキュメントを参照してください。

スーパーバイザー からの Tanzu Mission Control エージェントのアンインストール

スーパーバイザー から Tanzu Mission Control エージェントをアンインストールするには、vSphere with Tanzu におけるスーパーバイザー クラスタからのクラスタ エージェントの手動削除を参照してください。

Tanzu Mission Control Self-Managed の スーパーバイザー への登録

Tanzu Mission Control Self-Managed を スーパーバイザー に登録するには、TMC エージェントのカスタム リソース定義を作成して適用します。

Tanzu Mission Control Self-Managed について

インストール方法や構成方法など、Tanzu Mission Control Self-Managed の詳細については、ドキュメント「Installing and Running VMware Tanzu Mission Control Self-Managed」を参照してください。

Tanzu Mission Control Self-Managed の スーパーバイザー への登録

Tanzu Mission Control Self-Managed を スーパーバイザー に統合するには、TMC エージェントを参照するカスタム リソース定義を作成します。スーパーバイザー には、エージェントがインストールされている TMC の Kubernetes 名前空間が含まれています。

次の手順を実行します。

1 ドキュメントの説明に従って Tanzu Mission Control Self-Managed をインストールします。「Installing and Running VMware Tanzu Mission Control Self-Managed」を参照してください。

2 Web ブラウザを使用して、Tanzu Mission Control Self-Managed のローカル デプロイにアクセスします。

3 Tanzu Mission Control Self-Managed インストール環境のルート CA 証明書をエクスポートします。

- 既知の CA を使用している場合は、ブラウザのアドレス バーの左側にあるロック アイコンをクリックして、証明書を表示します。プライベート CA を使用している場合は、安全でないことを示すボタンをクリックして証明書を表示します。
- 証明書ダイアログのポップアップで、[`Details`] タブを選択してから [`Export `] ボタンを選択し、CA 証明書のコピーをダウンロードします。
- 任意のテキスト エディタを使用して CA 証明書ファイルを開き、CA 証明書の内容にアクセスします。

4 kubectl 向けの vSphere プラグイン を使用して、スーパーバイザー での認証を行います。

```
kubectl vsphere login --server=IP-ADDRESS --vsphere-username USERNAME
```

5 次のコマンドを実行して、利用可能な Kubectl コンテキストを一覧表示します。

```
kubectl config get-contexts
```

6 Tanzu Mission Control Self-Managed を実行中の TKG クラスタがプロビジョニングされているターゲットの vSphere 名前空間 にコンテキストを切り替えます。

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

7 次のコマンドを実行して、Kubernetes 名前空間を一覧表示します。

```
kubectl get ns
```

8 TMC の スーパーバイザー の Kubernetes 名前空間には、`svc-tmc-cXXXX` という名前が付けられます (XXXX は数字)。たとえば、`svc-tmc-c1208` など。この Kubernetes 名前空間があり、有効であることを確認します。

9 テキスト エディタを使用して、`agentconfig.yaml` という名前のカスタム リソース定義を作成します。このファイルには、TMC 名前空間、TMC Self-Managed デプロイのホスト名、および CA 証明書の内容が含まれます。

- `namespace` フィールドに TMC の Kubernetes 名前空間の名前を入力します。
- `caCerts` フィールドに CA 証明書を入力します。

- `allowedHostNames` フィールドに TMC ホスト名を入力します。

```
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 作成した `AgentConfig` yaml ファイルを適用します。

```
kubectl apply -f agentconfig.yaml
```

11 Tanzu Mission Control Self-Managed インストール環境の Web インターフェイスを使用して、登録プロセスを完了します。手順については、Tanzu Mission Control のドキュメントを参照してください。

TKG サービス クラスタとワークロードのバックアップとリストア 20

TKG サービス クラスタとワークロードをバックアップおよびリストアするには、このセクションを参照してください。

次のトピックを参照してください。

- TKG サービス クラスタとワークロードのバックアップとリストアに関する考慮事項
- Velero Plugin for vSphere を使用したワークロードのバックアップとリストア
- スタンドアローンの Velero と Restic を使用した スーパーバイザー での TKG クラスタ ワークロードのバックアップとリストア
- Velero と CSI スナップショットを使用したバックアップとリストア

TKG サービス クラスタとワークロードのバックアップとリストアに関する考慮事項

このトピックでは、TKG サービス クラスタで実行されているワークロードをバックアップおよびリストアするための考慮事項について説明します。

TKG サービス クラスタのバックアップとリストア

TKG クラスタをバックアップおよびリストアするには、スーパーバイザー データベースをバックアップします。これにより、vSphere 名前空間 オブジェクトと TKG クラスタ ノード仮想マシンをリストアできます。

vCenter Server 管理インターフェイスを通じて使用可能な vCenter Server バックアップ機能を使用して、スーパーバイザー のバックアップとリストアを有効にします。詳細については、vSphere IaaS control plane のバックアップとリストアに関するドキュメントを参照してください。

注： スーパーバイザー バックアップは、TKG クラスタ ノード仮想マシンをリストアする場合にのみ使用できます。スーパーバイザー バックアップを使用して、TKG クラスタにデプロイされたワークロードをリストアすることはできません。ワークロードは別途バックアップし、クラスタのリストア後にリストアする必要があります。

TKG サービス クラスタで実行されているワークロードのバックアップとリストア

表に、TKG クラスタで実行されているステートレスおよびステートフル ワークロードをバックアップおよびリストアするためのオプションの概要を示します。

注： Kubernetes クラスタをバックアップおよびリストアするときにスタンドアローンの Velero を使用すると、移植性が得られます。つまり、TKG サービス でプロビジョニングされていない Kubernetes クラスタにクラスタ ワークロードをリストアできるようにする場合は、スタンドアローンの Velero を使用することを検討してください。

使用例	ツール	コメント
スーパーバイザー上の TKG クラスタでステートレス ワークロードおよびステートフル ワークロードをバックアップし、スーパーバイザー上の TKG クラスタにリストアします。	Velero Plugin for vSphere Velero Plugin for vSphere を使用したワークロードのバックアップとリストアを参照してください。	Kubernetes メタデータとパーシステント ボリュームの両方をバックアップおよびリストアできます。 Velero スナップショット作成は、ステートフル アプリケーションでパーシステント ボリュームに使用されます。 Velero Plugin for vSphere もスーパーバイザーにインストールされ、構成されている必要があります。
スーパーバイザー上の TKG クラスタでステートレスおよびステートフル ワークロードをバックアップし、適合する Kubernetes クラスタにリストアします。	スタンドアローン Velero と Restic 『スタンドアローンの Velero と Restic を使用した スーパーバイザー での TKG クラスタ ワークロードのバックアップとリストア』を参照してください 。	Kubernetes メタデータとパーシステント ボリュームの両方をバックアップおよびリストアできます。 Restic は、ステートフル アプリケーションでパーシステント ボリュームに使用されます。 移植性が必要な場合は、このアプローチを使用します。
スーパーバイザー上の TKG クラスタでステートレスおよびステートフル ワークロードをバックアップし、適合する Kubernetes クラスタにリストアします。	CSI スナップショットを使用したスタンドアローン Velero 『Velero と CSI スナップショットを使用したバックアップとリストア』を参照してください	vSphere 8.0 には、vSphere 8.0 U2 以降および TKr v1.26 以降が必要です。

Velero Plugin for vSphere を使用したワークロードのバックアップとリストア

このセクションでは、Velero Plugin for vSphere を使用してクラスタ ワークロードをバックアップおよびリストアする方法について説明します。

TKG クラスタでの Velero Plugin for vSphere のインストールと構成

Velero Plugin for vSphere を使用して TKGS クラスタで実行されているワークロードをバックアップおよびリストアするには、このクラスタに Velero Plugin for vSphere をインストールします。

概要

Velero Plugin for vSphere は、TKGS クラスタのワークロードをバックアップおよびリストアするためのソリューションを提供します。パーシステント ワークロードの場合は、Velero Plugin for vSphere でパーシステント ボリュームのスナップショットを作成できます。

注： バックアップおよびリストアする TKGS クラスタ ワークロードに可搬性が必要な場合は、Velero Plugin for vSphere を使用しないでください。Kubernetes クラスタ間で可搬性を確保するには、スタンドアローン Velero と Restic を併用します。

前提条件：スーパーバイザー への Velero Plugin for vSphere のインストール

TKGS クラスタに Velero Plugin for vSphere をインストールするには、スーパーバイザー に Velero Plugin for vSphere がインストールされている必要があります。また、スーパーバイザー に NSX ネットワークが構成されている必要があります。#unique_22 を参照してください。

手順 1：Linux Workstation への Velero CLI のインストール

Velero CLI は、Velero とのインターフェイスを提供する標準ツールです。Velero CLI は Velero Plugin for vSphere CLI (`velero-vsphere`) よりも高機能で、Tanzu Kubernetes クラスタ ワークロードのバックアップとリストアに必要です。

Linux ワークステーションに Velero CLI をインストールします。理想的なのは、このワークステーションが、vSphere IaaS control plane 環境に関連付けられた CLI（`kubectl`、`kubectl-vsphere`、`velero-vsphere` など）を実行するジャンプ ホストと同じであることです。

Velero バージョン番号は `X.Y.Z` として表されます。使用するバージョンについては、Velero 互換性マトリックスを参照し、コマンドの実行時に適宜置き換えてください。

次の手順を実行して、Velero CLI をインストールします。

1 次のコマンドを実行します。

```
$ 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 Velero CLI がインストールされていることを確認します。

```
velero version

Client:
    Version: vX.Y.Z
```

手順 2：S3 互換バケットの詳細の取得

便宜上、この手順では、スーパーバイザー に Velero Plugin for vSphere をインストールするときに構成したのと同じ S3 互換オブジェクト ストアを使用していると仮定します。本番環境では、別のオブジェクト ストアを作成できます。

Velero Plugin for vSphere をインストールするには、S3 互換オブジェクト ストアに関する次の情報を指定する必要があります。

データ項目	値の例
s3Url	`http://my-s3-store.example.com`
aws_access_key_id	ACCESS-KEY-ID-STRING
aws_secret_access_key	SECRET-ACCESS-KEY-STRING

次の情報を使用して、`s3-credentials` という名前のシークレット ファイルを作成します。このファイルは、Velero Plugin for vSphere をインストールするときに参照します。

```
aws_access_key_id = ACCESS-KEY-ID-STRING
aws_secret_access_key = SECRET-ACCESS-KEY-STRING
```

手順 3（オプション A）：ラベルを使用した TKG クラスタへの Velero Plugin for vSphere のインストール（新しい方法）

vSphere 8 Update 3 以降を使用している場合は、ラベルを追加して TKG クラスタに Velero Plugin for vSphere を自動的にインストールできます。

1 バックアップ ストレージの場所にアクセス可能であることを確認します。

2 Velero vSphere Operator Core スーパーバイザー サービス が有効になっていることを確認します。

```
kubectl get ns | grep velero
svc-velero-domain-c9                        Active   18d
```

3 `velero` という名前の Kubernetes 名前空間が スーパーバイザー に作成されていることを確認します。

```
kubectl get ns | grep velero
svc-velero-domain-c9                        Active   18d
velero                                      Active   1s
```

4 Velero Plugin for vSphere スーパーバイザー サービス が スーパーバイザー で有効になっていることを確認します。

```
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 velero ラベルをクラスタに追加して、ターゲット TKG クラスタに対して Velero を有効にします。

```
kubectl label cluster CLUSTER-NAME --namespace CLUSTER-NS velero.vsphere.vmware.com/
enabled=true
```

注： この作業は、クラスタのプロビジョニング時に vSphere 名前空間 から実行します。

6 Velero がインストールされていること、およびクラスタの準備ができていることを確認します。

```
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
```

手順 3（オプション B）：手動による TKG クラスタへの Velero Plugin for vSphere のインストール（従来の方法）

Velero CLI を使用して、バックアップおよびリストアするターゲットの TKG クラスタに Velero Plugin for vSphere をインストールします。

Velero CLI コンテキストは、`kubectl` コンテキストに自動的に従います。Velero CLI コマンドを実行してターゲット クラスタに Velero と Velero Plugin for vSphere をインストールする前に、`kubectl` コンテキストをターゲット クラスタに設定してください。

1 kubectl 向けの vSphere プラグイン を使用して、スーパーバイザー での認証を行います。

2 `kubectl` コンテキストをターゲットの TKG クラスタに設定します。

```
kubectl config use-context TARGET-TANZU-KUBERNETES-CLUSTER
```

3 TKG クラスタで、Velero プラグイン `velero-vsphere-plugin-config.yaml` の構成マップを作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: velero-vsphere-plugin-config
data:
  cluster_flavor: GUEST
```

TKG クラスタに構成マップを適用します。

```
kubectl apply -n <velero-namespace> -f velero-vsphere-plugin-config.yaml
```

構成マップをインストールしなかった場合は、Velero Plugin for vSphere をインストールするときに次のエラーが表示されます。

```
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 次の Velero CLI コマンドを実行して、ターゲット クラスタに Velero をインストールします。

BUCKET-NAME、**REGION**（2 つのインスタンス）、および **s3Url** フィールドのプレースホルダ値を適切な値に置き換えます。前述のいずれかの手順に従わなかった場合は、シークレット ファイルの名前や場所、手動で作成した `velero` 名前空間の名前などの値を調整します。

```
./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 ターゲット クラスタに Velero Plugin for vSphere をインストールします。インストールされている Velero は Kubernetes API サーバと通信して、プラグインをインストールします。

```
velero plugin add vsphereveleroplugin/velero-plugin-for-vsphere:vX.Y.Z
```

補足：TKG クラスタからの Velero Plugin for vSphere のアンインストール

次の手順を実行して、Velero Plugin for vSphere をアンインストールします。

1 `kubectl` コンテキストをターゲット Tanzu Kubernetes クラスタに設定します。

```
kubectl config use-context TARGET-TANZU-KUBERNETES-CLUSTER
```

2 プラグインをアンインストールするには、次のコマンドを実行して、Velero のデプロイから velero-plugin-for-vsphere の InitContainer を削除します。

```
velero plugin remove vsphereveleroplugin/velero-plugin-for-vsphere:vX.Y.Z
```

3 プロセスを完了するには、バックアップ ドライバのデプロイと関連する CRD を削除します。

```
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
```

Velero Plugin for vSphere を使用した TKG クラスタ ワークロードのバックアップとリストア

Velero Plugin for vSphere を使用して、スーパーバイザー上の TKG クラスタで実行されているワークロードをバックアップおよびリストアできます。

前提条件

Velero Plugin for vSphere を使用して TKG クラスタ ワークロードをバックアップおよびリストアするには、最初に Velero Plugin for vSphere をターゲット クラスタにインストールする必要があります。TKG クラスタでの Velero Plugin for vSphere のインストールと構成を参照してください。

ワークロードのバックアップ

Velero バックアップを作成するコマンドの例を次に示します。

```
velero backup create <backup name> --include-namespaces=my-namespace
```

Velero バックアップは、すべてのローカル スナップショットが作成され、ボリューム スナップショットを除く Kubernetes メタデータがオブジェクト ストアにアップロードされた後に、`Completed` としてマークされます。この時点でも、非同期のデータ移動タスク、つまりボリューム スナップショットのアップロードはバックグラウンドで実行されており、完了に時間がかかることがあります。ボリューム スナップショットのステータスは、スナップショット カスタム リソース (CR) を監視することにより確認できます。

スナップショット

スナップショットは、パーシステント ボリュームをバックアップするために使用されます。スナップショット CR は、スナップショットが作成されたパーシステント ボリューム要求 (PVC) と同じ名前空間に、ボリューム スナップショットごとに作成されます。

PVC 名前空間のすべてのスナップショットを取得するには、次のコマンドを実行します。

```
kubectl get -n <pvc namespace> snapshot
```

スナップショットのカスタム リソース定義 (CRD) には、`.status.phase` フィールドのフェーズが多数あります。次に例を示します。

スナップショット フェーズ	説明
新規	まだ処理されていません
Snapshotted	ローカル スナップショットが作成されました
SnapshotFailed	ローカル スナップショットの作成に失敗しました
アップロードしています	スナップショットをアップロードしています
Uploaded	スナップショットがアップロードされました
UploadFailed	スナップショットのアップロードに失敗しました
Canceling	スナップショットのアップロードをキャンセルしています
キャンセルされました	スナップショットのアップロードがキャンセルされました
CleanupAfterUploadFailed	スナップショットのアップロード後のローカル スナップショットのクリーンアップに失敗しました

ワークロードのリストア

Velero リストアのコマンドの例を次に示します。

```
velero restore create --from-backup <velero-backup-name>
```

Velero リストアが `Completed` としてマークされるのは、ボリューム スナップショットと他の Kubernetes メタデータが現在のクラスタに正常にリストアされたときです。この時点で、このリストアに関連する vSphere プラグインのすべてのタスクも完了します。Velero バックアップの場合とは異なり、バックグラウンドの非同期のデータ移動タスクはありません。

CloneFromSnapshot

各ボリューム スナップショットからリストアするために、最初にスナップショットが作成された PVC と同じ名前空間に `CloneFromSnapshot` カスタム リソース (CR) が作成されます。PVC 名前空間のすべての CloneFromSnapshot を取得するには、次のコマンドを実行します。

```
kubectl -n <pvc namespace> get clonefromsnapshot
```

CloneFromSnapshot CRD には、`.status.phase` フィールドの主要なフェーズがいくつかあります。

スナップショット フェーズ	説明
新規	スナップショットからのクローン作成が完了していません
InProgress	vSphere ボリュームのスナップショットがリモート リポジトリからダウンロードされています
完了	スナップショットからのクローン作成が完了しました
Failed	スナップショットからのクローン作成に失敗しました

スタンドアローンの Velero と Restic を使用した スーパーバイザーでの TKG クラスタ ワークロードのバックアップとリストア

このセクションでは、スタンドアローンの Velero と Restic を使用して スーパーバイザー で実行されている TKG クラスタ ワークロードをバックアップおよびリストアする方法について説明します。

TKG クラスタへのスタンドアローン Velero と Restic のインストールと構成

スーパーバイザー上の TKG クラスタで実行中のワークロードをバックアップおよびリストアするには、データストアを作成し、Kubernetes クラスタに Velero と Restic をインストールします。

概要

TKG クラスタは仮想マシン ノードで実行されます。TKG クラスタ ワークロードをバックアップおよびリストアするには、そのクラスタに Velero と Restic をインストールします。

前提条件

Tanzu Kubernetes クラスタで実行されているワークロードをバックアップおよびリストアするために Velero と Restic をインストールするための次の前提条件を環境が満たしていることを確認します。

- いくつかのワークロード バックアップを保存するために十分なストレージを持つ Linux 仮想マシン。この仮想マシンに MinIO をインストールします。
- kubectl 向けの vSphere プラグイン と kubectl を含む vSphere 向け Kubernetes CLI Tools がインストールされる Linux 仮想マシン。このクライアント仮想マシンに Velero CLI をインストールします。このような仮想マシンがない場合は Velero CLI をローカルにインストールできますが、条件に合わせてインストール手順を調整する必要があります。
- Kubernetes 環境がインターネットにアクセスでき、クライアント仮想マシンからアクセスできること。

MinIO オブジェクト ストアのインストールと構成

Velero は、Kubernetes ワークロードのバックアップ先として S3 互換のオブジェクト ストアを必要とします。Velero は、そのようなオブジェクト ストア プロバイダをいくつかサポートしています。簡単にするために、これらの手順では、オブジェクト ストア仮想マシンでローカルに実行される S3 互換のストレージ サービスである MinIO を使用します。

1 MinIO をインストールします。

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
```

2 MinIO に実行権限を付与します。

```
chmod +x minio
```

3 ファイル システムに MinIO 用のディレクトリを作成します。

```
mkdir /DATA-MINIO
```

4 MinIO サーバを起動します。

```
./minio server /DATA-MINIO
```

5 MinIO サーバが起動すると、エンドポイント URL、AccessKey、SecretKey などの重要なデータストア インスタンスの詳細が得られます。表に示されているエンドポイント URL、AccessKey、SecretKey を記録します。

データストア メタデータ	値
エンドポイント URL	
AccessKey	
SecretKey	

6 ブラウザで MinIO サーバのエンドポイント URL を指定して、MinIO データストアを参照します。

7 MinIO サーバにログインし、AccessKey と SecretKey を指定します。

8 MinIO をサービスとして有効にするには、`minio.service` スクリプトをダウンロードして、MinIO に自動起動を構成します。

```
curl -O https://raw.githubusercontent.com/minio/minio-service/master/linux-systemd/
minio.service
```

9 `minio.service` スクリプトを編集し、`ExecStart` に次の値を追加します。

```
ExecStart=/usr/local/bin/minio server /DATA-MINIO path
```

10 変更したスクリプトを保存します。

11 次のコマンドを実行して MinIO サービスを構成します。

```
cp minio.service /etc/systemd/system
cp minio /usr/local/bin/
systemctl daemon-reload
systemctl start minio
systemctl status minio
systemctl enable minio
```

12 MinIO ブラウザを起動し、オブジェクト ストアにログインして、バックアップとリストア用の MinIO バケットを作成します。

13 バケット作成のアイコンをクリックします。

14 たとえば `my-cluster-backups` のように、バケット名を入力します。

15 バケットが作成されたことを確認します。

16 デフォルトでは、新しい MinIO バケットは読み取り専用です。Velero スタンドアローンのバックアップとリストアの場合、MinIO バケットは読み取り/書き込みである必要があります。バケットを読み取り/書き込みに設定するには、バケットを選択し、省略記号（点が並んだ記号）のリンクをクリックします。

17 [ポリシーの編集] を選択します。

18 ポリシーを [読み取りおよび書き込み] に変更します。

19 [追加] をクリックします。

20 ダイアログ ボックスを閉じるために、X をクリックします。

Velero CLI のインストール

仮想マシン クライアントまたはローカル マシンに Velero CLI をインストールします。

このドキュメントで使用されるバージョンは、*Tanzu Kubernetes Grid 2.2.0 用 Velero 1.9.7* です。

1 VMware Customer Connect ポータルの Tanzu Kubernetes Grid 製品ダウンロード ページから Velero をダウンロードします。

注： VMware からサポートを受けるには、VMware によって署名された Velero バイナリを使用する必要があります。

2 コマンド ラインを開き、Velero CLI をダウンロードしたディレクトリに移動します。

3 ダウンロード ファイルを解凍します。例：

```
gunzip velero-linux-vX.X.X_vmware.1.gz
```

4 Velero バイナリを確認します。

```
ls -l
```

5 Velero CLI に実行権限を付与します。

```
chmod +x velero-linux-vX.X.X_vmware.1
```

6 Velero CLI をシステム パスに移動して、グローバルに使用可能にします。

```
cp velero-linux-vX.X.X_vmware.1 /usr/local/bin/velero
```

7 インストールを確認します。

```
velero version
```

Tanzu Kubernetes クラスタへの Velero と Restic のインストール

Velero CLI のコンテキストは、kubectl のコンテキストに自動的に従います。Velero CLI コマンドを実行して Velero と Restic をターゲット クラスタにインストールする前に、kubectl のコンテキストを設定します。

1 MinIO バケットの名前を取得します。たとえば、`my-cluster-backups` です。

2 MinIO バケットの AccessKey と SecretKey を取得します。

3 Velero CLI の動作対象となるクラスタが認識されるように、コンテキストをターゲットの Kubernetes クラスタに設定します。

```
kubectl config use-context tkgs-cluster-name
```

4 `credentials-minio` という名前のシークレット ファイルを作成します。収集した MinIO サーバ アクセス認証情報を使用して、ファイルを更新します。例：

```
aws_access_key_id = 0XXNO8JCCGV41QZBV0RQ
aws_secret_access_key = clZ1bf8Ljkvkmq7fHucrKCkxV39BRbcycGeXQDfx
```

注： 「バックアップ ストアの取得中にエラーが発生しました」というエラー メッセージと「NoCredentialProviders: チェーン内に有効なプロバイダがありません」という説明が表示された場合は、認証情報ファイルの先頭に `[default]` 行を追加します。例：

```
[default]
aws_access_key_id = 0XXNO8JCCGV41QZBV0RQ
aws_secret_access_key = clZ1bf8Ljkvkmq7fHucrKCkxV39BRbcycGeXQDfx
```

5 ファイルを保存し、ファイルが正しい場所に置かれたことを確認します。

```
ls
```

6 次のコマンドを実行して、ターゲット Kubernetes クラスタに Velero と Restic をインストールします。両方の URL を MinIO インスタンスの URL に置き換えます。

```
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 Velero と Restic のインストールを確認します。

```
kubectl logs deployment/velero -n velero
```

8 `velero` 名前空間を確認します。

```
kubectl get ns
```

9 `velero` ポッドと `restic` ポッドを確認します。

```
kubectl get all -n velero
```

Restic DaemonSet のトラブルシューティング（必要な場合）

Kubernetes クラスタで 3 ポッドの Restic DaemonSet を実行するには、Restic DaemonSet の仕様を更新して hostPath を変更することが必要な場合があります。この問題の詳細については、Velero のドキュメントで Restic Integration を参照してください。

1 3 ポッドの Restic DaemonSet を確認します。

```
kubectl get pod -n velero
```

ポッドのステータスが CrashLoopBackOff の場合は、次のように編集します。

2 `edit` コマンドを実行します。

```
kubectl edit daemonset restic -n velero
```

3 hostPath を `/var/lib/kubelet/pods` から `/var/vcap/data/kubelet/pods` に変更します。

```
- hostPath:
    path: /var/vcap/data/kubelet/pods
```

4 ファイルを保存します。

5 3 ポッドの Restic DaemonSet を確認します。

```
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
```

Velero のメモリ制限の調整（必要な場合）

Velero のバックアップが数時間に渡って `status=InProgress` を返す場合は、メモリ設定の limits と requests の値を増やします。

1 次のコマンドを実行します。

```
kubectl edit deployment/velero -n velero
```

2 メモリ設定の limits と requests をデフォルトの `256Mi` および `128Mi` から `512Mi` および `256Mi` に変更します。

```
ports:
- containerPort: 8085
  name: metrics
  protocol: TCP
resources:
  limits:
    cpu: "1"
    memory: 512Mi
  requests:
    cpu: 500m
    memory: 256Mi
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
```

スタンドアローンの Velero と Restic を使用したクラスタ ワークロードのバックアップとリストア

スタンドアローンの Velero と Restic を使用して、TKG クラスタで実行されているワークロードをバックアップおよびリストアできます。これは Velero Plugin for vSphere を使用する代わりの方法です。スタンドアローンの Velero は、主に移植性が必要な場合に利用します。ステートフル ワークロードには、Restic が必要です。

前提条件

スタンドアローンの Velero と Restic を使用して TKG クラスタでワークロードをバックアップおよびリストアするには、ターゲット クラスタにスタンドアローン バージョンの Velero と Restic をインストールする必要があります。リストアを別のターゲット クラスタで実行する場合は、ターゲット クラスタにも Velero と Restic をインストールする必要があります。TKG クラスタへのスタンドアローン Velero と Restic のインストールと構成を参照してください。

TKG クラスタで実行されているステートレス アプリケーションのバックアップ

TKG クラスタで実行されているステートレス アプリケーションをバックアップするには、Velero を使用する必要があります。

この例では、サンプル ステートレス アプリケーションの名前空間にすべてのアプリケーション コンポーネントがある場合に、`--include namespaces` タグを使用して、そのサンプル ステートレス アプリケーションをバックアップおよびリストアする方法を示します。

```
velero backup create example-backup --include-namespaces example-backup
```

次の内容が表示されます。

```
Backup request "example-backup" submitted successfully.
Run `velero backup describe example-backup` or `velero backup logs example-backup` for more
details.
```

作成されたバックアップを確認します。

```
velero backup get
```

```
velero backup describe example-backup
```

MinIO サーバなどの S3 互換オブジェクト ストアの Velero バケットを確認します。

Velero は Kubernetes カスタム リソース定義 (CRD) にメタデータを書き込みます。

```
kubectl get crd
```

Velero CRD では、次のような特定のコマンドを実行できます。

```
kubectl get backups.velero.io -n velero
```

```
kubectl describe backups.velero.io guestbook-backup -n velero
```

TKG クラスタで実行されているステートレス アプリケーションのリストア

TKG クラスタで実行されているステートレス アプリケーションをリストアするには、Velero を使用する必要があります。

サンプル アプリケーションのリストアをテストするために、サンプル アプリケーションを削除します。

名前空間を削除します。

```
kubectl delete ns guestbook
namespace "guestbook" deleted
```

アプリケーションをリストアします。

```
velero restore create --from-backup example-backup
```

次の内容が表示されます。

```
Restore request "example-backup-20200721145620" submitted successfully.
Run `velero restore describe example-backup-20200721145620` or `velero restore logs example-
backup-20200721145620` for more details.
```

アプリケーションがリストアされたことを確認します。

```
velero restore describe example-backup-20200721145620
```

検証のために、次のコマンドを実行します。

```
velero restore get
```

```
kubectl get ns
```

```
kubectl get pod -n example
```

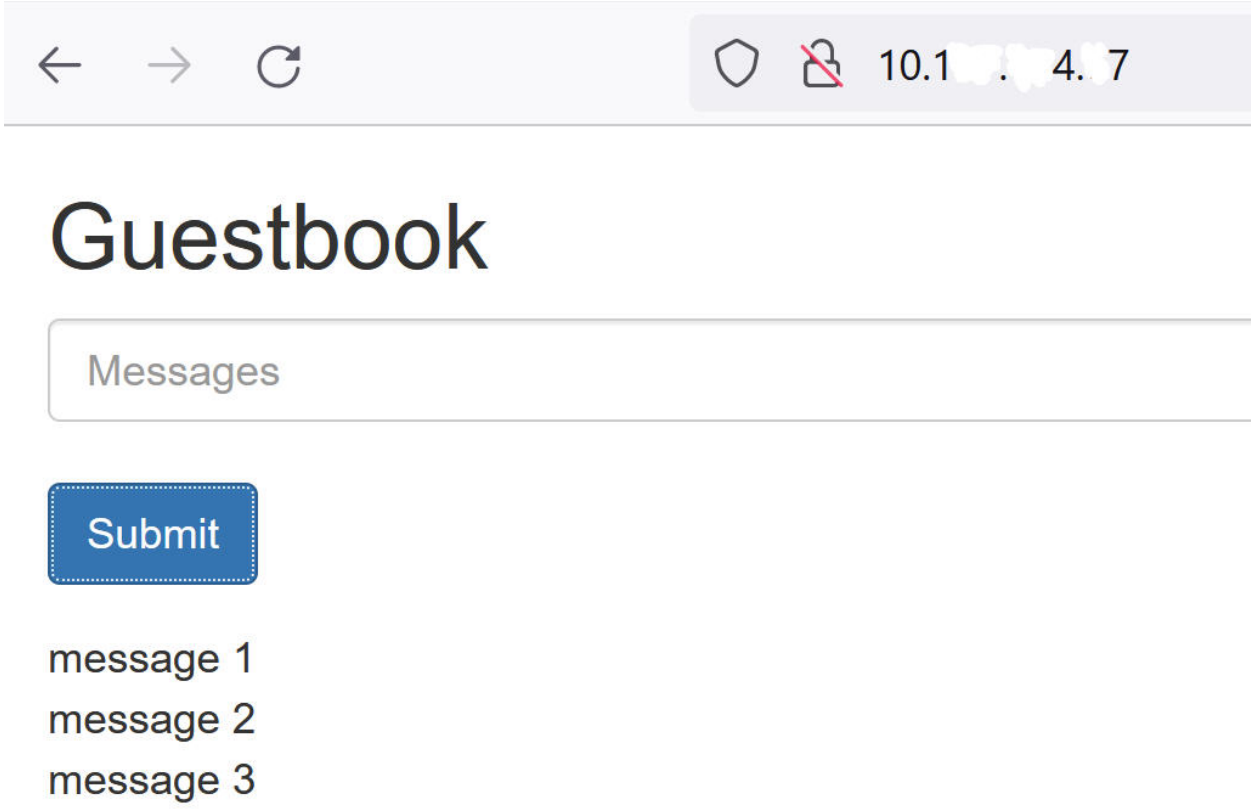
```
kubectl get svc -n example
```

TKG クラスタで実行されているステートフル アプリケーションのバックアップ

TKG クラスタで実行されているステートフル アプリケーションをバックアップする場合は、保存されているアプリケーション メタデータとアプリケーション データの両方をパーシステント ボリュームにバックアップする必要があります。これを行うには、Velero と Restic の両方が必要です。

この例では、Guestbook アプリケーションを使用します。Guestbook アプリケーションは、TKG クラスタにデプロイされているものと想定します。TKG クラスタへの Guestbook アプリケーションのデプロイを参照してください。

ステートフルなバックアップとリストアを説明するために、フロントエンド Web ページを使用してメッセージを Guestbook アプリケーションに送信し、そのメッセージが保存されるようにしてみましょう。例：



この例では、`--include namespace` タグやポッドの注釈を使用して、Guestbook アプリケーションをバックアップおよびリストアする方法を示します。

注： この例では、注釈を使用します。ただし、Velero バージョン 1.5 以降では、注釈は不要になりました。注釈を使用しないようにするには、バックアップの作成時に `--default-volumes-to-restic` オプションを使用してください。これにより、すべての PV が Restic を使用して自動的にバックアップされます。詳細については https://velero.io/docs/v1.5/restic/を参照してください。

バックアップ手順を開始するために、ポッドの名前を取得します。

```
kubectl get pod -n guestbook
```

例：

```
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
```

パーシステント ボリュームが Redis ポッドに接続されます。これらのステートフル ポッドは、Restic を使用してバックアップします。このため、volumeMount の名前を使用して、ステートフル ポッドに注釈を追加する必要があります。

ステートフル ポッドに注釈を付けるには、volumeMount を知る必要があります。mountName を取得するには、次のコマンドを実行します。

```
kubectl describe pod redis-leader-deployment-64fb8775bf-kbs6s -n guestbook
```

結果には、redis-leader-data からの Containers.leader.Mounts: /data が表示されます。この最後のトークンが、リーダー ポッドの注釈に使用する volumeMount 名です。フォロワーの場合は、redis-follower-data になります。volumeMount 名は、ソース YAML から取得することもできます。

各 Redis ポッドに注釈を付けます。次に例を示します。

```
kubectl -n guestbook annotate pod redis-leader-64fb8775bf-kbs6s backup.velero.io/backup-
volumes=redis-leader-data
```

次の内容が表示されます。

```
pod/redis-leader-64fb8775bf-kbs6s annotated
```

注釈を確認します。

```
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
```

Velero バックアップを実行します。

```
velero backup create guestbook-backup --include-namespaces guestbook
```

次の内容が表示されます。

```
Backup request "guestbook-backup" submitted successfully.
Run `velero backup describe guestbook-pv-backup` or `velero backup logs guestbook-pv-backup`
for more details.
```

作成されたバックアップを確認します。

```
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>
```

バックアップの詳細を確認します。

```
velero backup describe guestbook-backup --details
```

Velero では、他にも次のようなコマンドを実行できます。

```
kubectl get backups.velero.io -n velero

NAME               AGE
guestbook-backup   4m58s
```

次のコマンドも実行できます。

```
kubectl describe backups.velero.io guestbook-backup -n velero
```

TKG 2.0 クラスタで実行されているステートフル アプリケーションのリストア

TKG クラスタで実行されるステートフル アプリケーションをリストアする場合は、保存されているアプリケーション メタデータとアプリケーション データの両方をパーシステント ボリュームにリストアする必要があります。これを行うには、Velero と Restic の両方が必要です。

この例では、前のセクションの説明に従ってステートフル Guestbook アプリケーションをバックアップしたものと想定しています。

ステートフル アプリケーションのリストアをテストするために、その名前空間を削除します。

```
kubectl delete ns guestbook
namespace "guestbook" deleted
```

アプリケーションが削除されたことを確認します。

```
kubectl get ns
kubectl get pvc,pv --all-namespaces
```

バックアップからアプリケーションをリストアするには、次のコマンド構文を使用します。

```
velero restore create --from-backup <velero-backup-name>
```

例：

```
velero restore create --from-backup guestbook-backup
```

次のようなメッセージが表示されます。

```
Restore request "guestbook-backup-20200723161841" submitted successfully.
Run `velero restore describe guestbook-backup-20200723161841` or `velero restore logs
guestbook-backup-20200723161841` for more details.
```

ステートフル Guestbook アプリケーションがリストアされたことを確認します。

```
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
```

さらに次のコマンドを実行して、リストアを確認します。

```
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>
```

名前空間がリストアされたことを確認します。

```
kubectl get ns

NAME              STATUS   AGE
default           Active   16d
guestbook         Active   76s
...
velero            Active   2d2h
```

アプリケーションがリストアされたことを確認します。

```
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
```

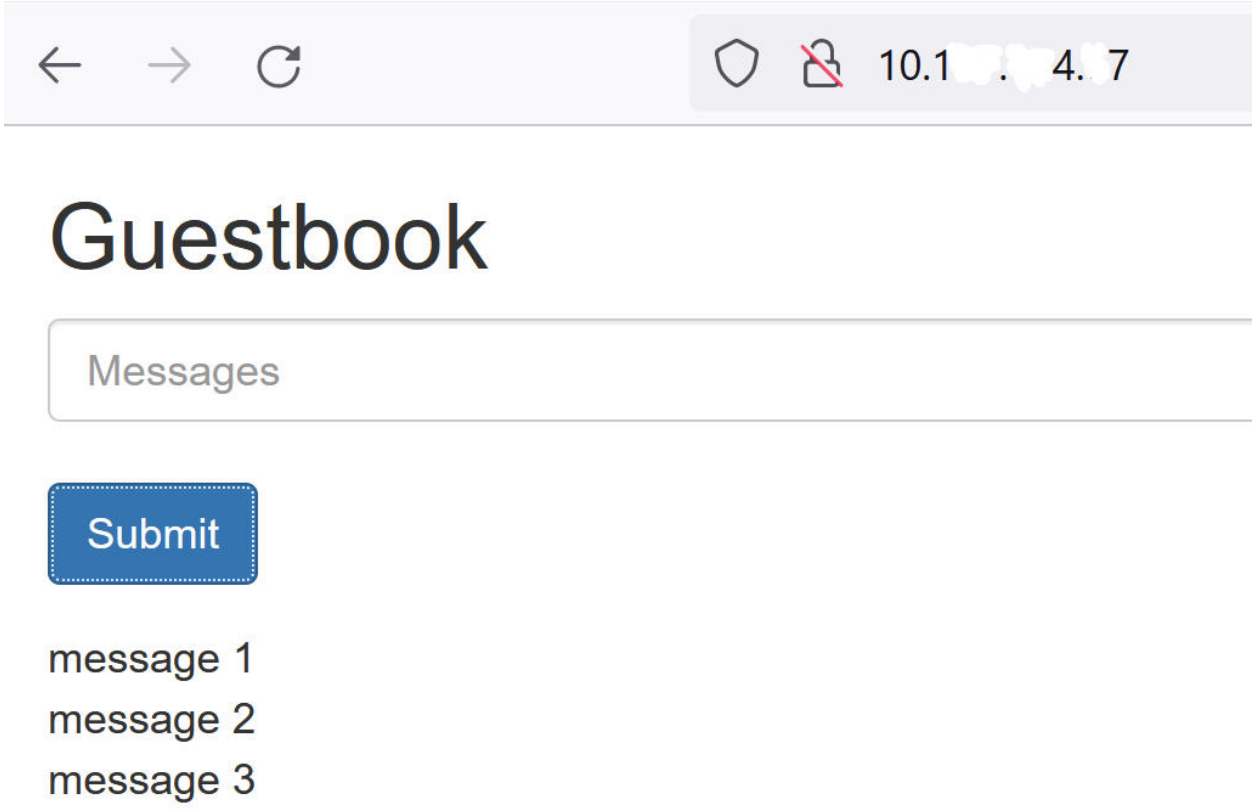
パーシステント ボリュームがリストアされたことを確認します。

```
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
```

最後に、guestbook-frontend サービスの外部 IP アドレスを使用して Guestbook フロントエンドにアクセスし、チュートリアルの最初に送信したメッセージがリストアされたことを確認します。例：



Velero と CSI スナップショットを使用したバックアップとリストア

Velero と CSI スナップショットを使用して、スーパーバイザー にプロビジョニングされた TKG クラスタで実行されているワークロードに関して CSI で作成されたパーシステント ボリュームをバックアップおよびリストアできます。

要件

次の要件を満たす必要があります。

- vSphere 8.0 U2 以降
- vSphere 8.x 以降の場合、Tanzu Kubernetes リリース v1.26.5
- ボリューム スナップショットをサポートする CSI ドライバを使用して作成されたパーシステント ボリューム

注目: Velero と CSI スナップショットを使用できるのは、ボリューム スナップショットをサポートする CSI ドライバを使用して作成されたパーシステント ボリュームの場合のみです。詳細については、『vSphere IaaS 制御プレーンでの TKG サービスの使用』の「TKG クラスタでのスナップショットの作成」を参照してください。

手順

Velero とコンテナ ストレージ インターフェイス (CSI) スナップショットを使用して、TKGS クラスタで実行されているワークロードをバックアップおよびリストアできます。Velero ノードエージェントは、CSI スナップショットのデータ移動を使用してバックアップとリストアの具体的なタスクを実行するモジュールをホストする DaemonSet です。詳細については、「Container Storage Interface Snapshot Support in Velero」を参照してください。

1 MinIO や AWS S3 バケットなどの S3 互換ストレージの場所を作成します。

次の例では、AWS S3 バケットを使用しています。

MinIO を使用するには、「MinIO オブジェクト ストアのインストールと構成」を参照してください。

2 kubectl を実行しているクラスタ クライアントに Velero CLI をインストールします。

https://github.com/vmware-tanzu/velero/releases からダウンロードします。

次のいずれかのリンクからインストール手順を参照してください。

- 手順 1：Linux Workstation への Velero CLI のインストール
- Velero CLI のインストール
- https://velero.io/docs/v1.12/basic-install/#install-the-cli

3 Velero バックアップを実行する TKG サービス クラスタに接続します。

Kubectl を使用した vCenter Single Sign-On ユーザーとしての TKG サービス クラスタへの接続を参照してください。

4 たとえば、AWS S3 ストレージとそれに対応する認証情報ファイルを指定して Velero インストール コマンドを実行します。

```
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
```

注： Velero v1.14 リリース以降、Velero CSI プラグインは Velero とマージされています。したがって、Velero v1.14 以降をインストールする場合、Velero CSI プラグインをインストールする必要はありません。した場合は、Velero ポッドの起動に失敗します。

TKG サービス クラスタのトラブルシューティング

21

クラスタ ノードへの接続、クラスタ リソース階層の表示、ログ ファイルの収集など、さまざまな方法を使用して、TKG サービス クラスタのトラブルシューティングを行うことができます。

次のトピックを参照してください。

スーパーバイザー 上の TKG クラスタに対するトラブルシューティング実行時のログの取得

スーパーバイザー サポート バンドル、ワークロード管理ログ、CAPI、CAPV、仮想マシン オペレータ、TKG コントローラ マネージャのログなど、スーパーバイザー 上の TKG クラスタのトラブルシューティングに関するさまざまなログを取得するには、このトピックを参照してください。

スーパーバイザー のサポート バンドルの収集

TKG クラスタ エラーをトラブルシューティングするために、スーパーバイザー ログをエクスポートできます。通常、これらのログの確認は、VMware のサポートと相談しながら行います。

1 vSphere Client を使用して、vSphere IaaS control plane 環境にログインします。

2 [メニュー] - [ワークロー管理] の順に選択します。

3 [スーパーバイザー] タブを選択します。

4 ターゲットの [スーパーバイザー] インスタンスを選択します。

5 [ログのエクスポート] を選択します。

サポート バンドルを収集したら、ナレッジベースの記事「Secure FTP ポータル経由での VMware への診断情報のアップロード」(http://kb.vmware.com/kb/2069559) を参照してください。「Gathering Logs for vSpehre with Tanzu」も参照してください。

TKG クラスタのサポート バンドルの収集

TKC Support Bundler ユーティリティを使用して、TKG クラスタのログ ファイルを収集し、問題のトラブルシューティングを実行できます。

TKC Support Bundler ユーティリティを入手して使用するには、VMware サポートのナレッジベースの記事「Gathering Logs for vSpehre with Tanzu」を参照してください。

ワークロード管理のログ ファイルのテール

ワークロード制御プレーン (WCP) ログ ファイルをテールすると、スーパーバイザー および TKG クラスタのエラーのトラブルシューティングに役立ちます。

1 vCenter Server Appliance への SSH 接続を確立します。

2 `root` ユーザーとしてログインします。

3 `shell` コマンドを実行します。

次のメッセージが表示されます。

```
Shell access is granted to root
root@localhost [ ~ ]#
```

4 次のコマンドを実行して、WCP ログ ファイルをテールします。

```
tail -f /var/log/vmware/wcp/wcpsvc.log
```

スーパーバイザー からの TKG 固有のログの収集

スーパーバイザー は、TKG 2.0 にインフラストラクチャを提供するいくつかの Kubernetes ポッドを実行します。

```
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
```

インフラストラクチャ ポッドとは、レプリカを実行する環境のことです。どのレプリカがリーダーであるかを判断し、そのログで最新の情報を確認する必要があります。リーダー以外は通常、リースの取得の試行に関するログを記録した後に停止します。

スーパーバイザー にログインし、スーパーバイザー vSphere 名前空間 を使用してこれらのポッドを確認する必要があります。

ラベル セレクタを使用するログは機能しないため、ポッド名の末尾に追加されるランダムな文字列を具体化する必要が生じる場合があります。grep 'error' または grep -i 'error' へ出力をパイプ処理することから始めると便利な場合があります。たとえば、`kubectl logs <args> | grep error` です。

CAPI ログ

クラスタ API プロバイダ：

```
kubectl logs -n vmware-system-capw -c manager vmware-system-capw-capi-controller-manager-
<id>
```

CAPV ログ

Cluster API for vSphere プロバイダ：

```
kubectl logs -n vmware-system-capv -c manager vmware-system-capw-v1alpha3-vmware-system-
capv-v1alpha3-controller-manager-<id>
```

仮想マシン オペレータのログ

仮想マシン オペレータ：

```
kubectl logs -n vmware-system-vmop -c manager vmware-system-vmop-controller-manager-<id>
```

TKG コントローラ マネージャのログ

GCM コントローラ マネージャ

```
kubectl logs -n vmware-system-tkg -c manager vmware-system-tkg-controller-manager-<id>
```

スーパーバイザー での TKG コンポーネントの健全性の確認

TKG コンポーネントに関して スーパーバイザー の健全性を確認するさまざまな方法については、このトピックを参照してください。

スーパーバイザー ポッドの状態の確認

スーパーバイザー ポッドは TKG インフラストラクチャ コンポーネントを実行します。

スーパーバイザー 上のすべてのポッドが実行状態であるかどうかを確認します。

```
kubectl get pods -A | grep "Running"
```

注： または、`grep -v "Running"` を使用して、実行中でないポッドを返すことができます。

例：

```
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
```

スーパーバイザー 上のいずれかのポッドが実行状態でない場合は、次のコマンドを使用してそのポッドを調べます。

```
kubectl describe pod <POD Name> -n <Namespace>
```

スーパーバイザー リソースの状態の確認

TKG コントローラのリソース：

```
kubectl get tkc
```

クラスタ API リソース（CAPI、CABPK、CAPW、CAPV）：

```
kubectl get cluster-api
```

仮想マシン オペレータのリソース：

```
kubectl get virtualmachines,virtualmachineservices,virtualmachinesetresourcepolicies
```

仮想マシン オペレータのリソース（クラスタ スコープ、コンテンツ ライブラリから同期）：

```
kubectl get virtualmachineimages
```

ストレージのリソース：

```
kubectl get persistentvolumeclaims,cnsnodevmattachment,cnsvolumemetadatas
```

ネットワークのリソース（NSX 固有）：

```
kubectl get service,lb,lbm,vnet,vnetif,nsxerrors,nsxnetworkinterfaces
```

スーパーバイザー のすべてのリソースの取得およびファイルへの書き込み：

```
kubectl api-resources --namespaced -o name | paste -d',' -s | xargs kubectl get -n
<namespace> > resources_in_namespace.txt
```

クラスタ API のデプロイが存在することを確認

CAPI、CAPW、CAPV のデプロイが存在することを確認します。

```
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
```

サポート バンドル ファイルの確認

スーパーバイザー のサポート バンドルの収集内の `commands/` フォルダには、WCP の起動プロセスで発生した内容の詳細を提供する journalctl ログがあります。

`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`

TKG クラスタの健全性の確認

すべてのクラスタ ノード（仮想マシン）が準備完了状態であることを確認します。

```
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
```

すべてのポッドが実行中であることを確認します。

```
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
```

TKG クラスタの状態を取得して説明します。

```
kubectl get tkc <clustername>
```

```
kubectl describe tkc <clustername>
```

TKG コントローラ マネージャの健全性の確認

TKG コントローラ マネージャの状態と健全性を確認します。

```
kubectl get deployments -n vmware-system-tkg vmware-system-tkg-controller-manager -o yaml
```

仮想マシン オペレータの健全性の確認

ポッドが実行中である必要があります。

```
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
```

仮想マシン オペレータは `VirtualNetworkInterface` を作成して、そのステータスを確認します。ノード仮想マシンが IP アドレスを取得しない場合は、その領域を最初に確認する必要があります。仮想マシンの作成はこのフェーズを通過しましたか。

仮想マシン オペレータは、`VirtualMachineService` の調整とステータスの更新も行います。TKG クラスタの Kubernetes API に外部 IP アドレスからアクセスできない場合は、仮想マシン オペレータのログを確認します。

たとえば、いずれかの仮想マシン オペレータ ポッドを選択し、名前空間を指定して、マネージャ コンテナを指定します。(`logs` コマンドはコンテナ用です。コントローラ ポッドの内部には、ログを確認できるマネージャ コンテナがあります)。

```
kubectl logs -f vmware-system-vmop-controller-manager-c8499b9df-5h6f9 -n vmware-system-vmop
manager
```

TKG クラスタ接続の問題とログイン エラーのトラブルシューティング

このセクションは、TKG クラスタ接続の問題とログイン エラーのトラブルシューティングを実行する際に使用します。

権限不足エラー

vSphere 名前空間 に関して必要な権限がない場合は、vCenter Single Sign-On ユーザーとして スーパーバイザー または TKG クラスタに接続できません。

vCenter Single Sign-On ユーザーとして スーパーバイザー または TKG クラスタに接続しようとすると、kubectl 向けの vSphere プラグイン から「`Error from server (Forbidden)`」というエラー メッセージが返されます。

vSphere 名前空間 に対する十分なロール権限がないか、ユーザー アカウントにアクセス権が付与されていません。

操作しているユーザーが、クラスタを運用している DevOps エンジニアである場合は、vSphere 名前空間 の 編集 権限が自分に付与されているか vSphere 管理者に確認してください。操作しているユーザーが、クラスタを使用してワークロードをデプロイしている開発者である場合は、クラスタへのアクセス権が自分に付与されているかクラスタ管理者に確認してください。

Kubectl vSphere ログイン エラー

kubectl 向けの vSphere プラグイン を使用して スーパーバイザー または TKG クラスタにログインしようとしたときに、次のエラーが表示される場合は、ログイン エラーが原因である可能性があります。

```
Failed to get available workloads, response from the server was invalid.
```

ログイン エラーをトラブルシューティングするには、`-v=10` を使用して、より詳細なログ出力を取得します。

```
kubectl vsphere login --server=10.110.150.56 --vsphere-username user@vsphere.local -v=10
```

たとえば、次に示すのは、詳細出力を使用して `invalid or missing credentials` エラーを表示したものです。

```
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.
```

スーパーバイザー への SSH 接続

ログイン エラーをトラブルシューティングするには、スーパーバイザー に SSH で接続することが必要な場合があります。

注意: スーパーバイザー 制御プレーン ノードに SSH 接続すると、スーパーバイザー クラスタを永続的に破損させる権限が付与されます。VMware のサポートは、ユーザーが スーパーバイザー 制御プレーン ノードから スーパーバイザー コンポーネントに変更を加えたという証拠を検出した場合、スーパーバイザー クラスタをサポート対象外としてマークし、vSphere IaaS control plane ソリューションの再デプロイをユーザーに求める可能性があります。このセッションは、ネットワークのテスト、ログの確認、kubectl logs/get/describe コマンドの実行にのみ使用してください。KB または VMware のサポートによる特別な許可がない場合は、このセッションから何らかの項目をデプロイ、削除、編集しないでください。

スーパーバイザー 制御プレーン ノードに SSH 接続するには、次の手順を実行します。

1 root ユーザー アカウントを使用して vCenter Server にログインします。

2 データセンター CLI を対話モードで使用するには、`dcli +i` と入力します。

3 コマンド `namespacemanagement software clusters list` を実行して、スーパーバイザー のステータスを取得します。

4 `exit` と入力して `dcli` シェルを終了します。

5 `shell` と入力して Bash シェル モードを開始します。

6 `/usr/lib/vmware-wcp/decyptK8Pwd.py` と入力して スーパーバイザー の IP アドレスとパスワードを取得します。

7 `ssh 10.100.150.56` と入力して SSH で スーパーバイザー に接続し、例の IP アドレスを前のコマンドで返された IP アドレスに置き換えます。

コンテンツ ライブラリ エラーのトラブルシューティング

Tanzu Kubernetes リリース コンテンツ ライブラリ エラーのトラブルシューティングについては、このトピックのヒントを参照してください。

TKR リソースが見つからない

vSphere 管理者がコンテンツ ライブラリを作成し、それをサポートされている Tanzu Kubernetes リリースと同期しました。TKG クラスタをデプロイする vSphere 名前空間 にコンテンツ ライブラリを割り当てました。スーパーバイザー にログインしており、コンテキストを vSphere 名前空間 に切り替えます。

次のコマンドを実行すると、`No resources found` が返されます。

```
kubectl get tanzukubernetesreleases
```

```
kubectl get tkr
```

トラブルシューティングを行うには、以下のコマンドを実行します。

```
kubectl get virtualmachineimages -A
```

```
kubectl get vmimage -o wide
```

コンテンツ ライブラリがあり、名前空間に登録されているかどうかを確認します。

```
kubectl get contentsources
```

```
 kubectl get contentsourcebindings -A
```

解決するには、vCenter Server 管理インターフェイスにログインします。[サービス] に移動し、[コンテンツ ライブラリ サービス] を選択し、[再起動] をクリックします。

これで問題が解決しない場合は、名前空間からコンテンツ ライブラリを削除しなければならないことがあります。そうするには、新しいコンテンツ ライブラリを作成して名前空間に追加し、前のコンテンツ ライブラリを削除します。

ライブラリ アイテムの取得に失敗する

TKG クラスタをプロビジョニングしようとしているときに、vSphere 名前空間 と同期され、関連付けられているサブスクライブ済みコンテンツ ライブラリからアイテムを取得できません。

次のエラーが表示されます。

```
Internal error occurred: get library items failed for.
```

サブスクライブ済みコンテンツ ライブラリがストレージ容量の制限に達した場合は、TKG クラスタをプロビジョニングできません。コンテンツ ライブラリは、接続されたストレージによってバッキングされます。時間の経過とともにリリースされる Kubernetes のバージョンが増えて、OVA ファイルがライブラリに同期されるため、ストレージ容量がいっぱいになることがあります。

TKR を自動的に同期している場合は、手動同期に切り替えて、必要な TKR イメージのみをローカルに保存することを検討してください。すでにオンデマンド同期を使用している場合は、不要になったイメージをライブラリから削除します。また、新しいコンテンツ ライブラリに移行することもできます。

ローカル コンテンツ ライブラリに TKR が見つからない

ローカル コンテンツ ライブラリは、インターネット制限環境で使用できます。

ローカル コンテンツ ライブラリを作成するときに、セキュリティ ポリシーをライブラリに適用するオプションがあります。Tanzu Kubernetes リリース がライブラリにアップロードしてある場合でも、次のいずれかの条件に該当する場合は、TKG クラスタでこのリリースを使用できません。

- コンテンツ ライブラリ内の OVF パッケージに署名がない。
- OVF パッケージが無効な証明書で署名されている。
- OVF パッケージは、ローカル コンテンツ ライブラリが構成されている vCenter Server によって信頼されていない証明書で署名されている。

OVA および VMDK ファイルをコンテンツ ライブラリにアップロードするときに、証明書ファイルがファイルのアップロード元のホーム ディレクトリにあることを確認します。

仮想マシン クラスのエラーのトラブルシューティング

仮想マシン クラスは、TKG サービス クラスタをプロビジョニングする vSphere 名前空間 に関連付ける必要があります。

仮想マシン クラス バインド エラー

ターゲット vSphere 名前空間 に追加していない 1 つ以上の仮想マシン クラスを使用して TKGS クラスタをプロビジョニングすると、`VirtualMachineClassBindingNotFound` エラーが表示されます。その例を下に示します。

```
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
```

エラーを修正するには、TKG サービス クラスタに使用する仮想マシン クラスを使用して vSphere 名前空間 を構成します。`kubectl get virtualmachineclass` コマンドを実行して、vSphere 名前空間 に関連付けられている仮想マシン クラスを表示します。

注意: `kubectl get virtualmachineclassbindings` コマンドは vSphere 8 U3 の時点で廃止されました。正しいコマンドは `virtualmachineclass` です。

注意: `kubectl get virtualmachineclasses` コマンドは、スーパーバイザー で使用できるすべての仮想マシン クラスを返します。ただし、クラスタのプロビジョニングに使用できるのはターゲット vSphere 名前空間 に関連付けられている仮想マシン クラスのみであるため、複数バージョンの名詞は情報を得る目的にのみ使用でき、プロビジョニング時に使用することはできません。

TKGS クラスタ プロビジョニング エラーのトラブルシューティング

TKGS クラスタをプロビジョニングできない場合は、この一般的なエラーのリストを確認してトラブルシューティングを行います。

クラスタ API ログの確認

TKG クラスタを作成できない場合は、CAPW/V が機能していることを確認します。

CAPW/V コントローラは、クラスタ API のインフラストラクチャ固有の実装です。CAPW/V を有効にするには、スーパーバイザー を使用します。CAPW/V は TKG のコンポーネントであり、TKG クラスタのライフサイクルを管理します。

CAPW/V は、VirtualNetwork の作成と更新を行います。VirtualNetwork の準備ができている場合にのみ、クラスタ ノードの作成を進めることができます。クラスタ作成ワークフローは、このフェーズを通過しましたか。

CAPW/V は、VirtualMachineService の作成と更新を行います。VirtualMachineService は正常に作成されましたか。外部 IP は取得されましたか。クラスタ作成ワークフローは、このフェーズを通過しましたか。

これらの質問に答えるには、次のようにクラスタ API のログを調べます。

```
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-...
```

クラスタ仕様の検証エラー

YAML 仕様に従って、キー名にスペース文字を使用できます。これはスペースを含むスカラー文字列であり、引用符は必要ありません。

ただし、TKGS の検証では、キー名にスペース文字を使用することはできません。TKGS では、有効なキー名は、英数字、ダッシュ（key-name など）、アンダースコア（KEY_NAME など）、ドット（key.name など）のみで構成する必要があります。

クラスタ仕様でキー名にスペース文字を使用すると、TKGS クラスタはデプロイされません。vmware-system-tkg-controller-manager ログには次のエラー メッセージが記録されます。

```
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]+')"
```

このエラーを修正するには、スペース文字を完全に削除するか、サポートされている文字に置き換えます。

TKG クラスタ YAML の適用時のエラー

TKG クラスタ YAML の適用時にエラーが発生した場合は、次のようにトラブルシューティングします。

クラスタ ネットワークが正しい状態でない

TKG クラスタのプロビジョニング ワークフローを理解します。

- CAPV は、各 TKG クラスタ ネットワークに対して VirtualNetwork オブジェクトを作成します。
- スーパーバイザー が NSX ネットワークを使用するように構成されている場合、NCP は VirtualNetwork オブジェクトを監視し、各 VirtualNetwork に対して NSX Tier-1 ルーターと NSX セグメントを作成します。
- CAPV は VirtualNetwork のステータスをチェックし、準備ができたらワークフローの次の手順に進みます。

仮想マシン サービス コントローラは、CAPV によって作成されたカスタム オブジェクトを監視し、これらの仕様を使用して、TKG クラスタを構築する仮想マシンを作成および構成します。

NSX Container Plugin (NCP) は、Kubernetes API を介して etcd に追加されたネットワーク リソースを監視し、NSX で対応するオブジェクトの作成を調整するコントローラです。

これらの各コントローラは、スーパーバイザー 制御プレーン上で Kubernetes ポッドとして実行されます。ネットワークの問題のトラブルシューティングを実行するには、CAPV コントローラ ログ、仮想マシン サービス ログ、および NCP ログを確認します。

コンテナ ログを確認します。`name-XXXX` は、実行したときの一意のポッド名です

```
kubectl get pods -A
kubectl logs pod/name-XXXXX -c pod-name -n namesapce
```

制御プレーン ノード数が無効

スーパーバイザー 上の TKG クラスタでは、1 つまたは 3 つの制御プレーン ノードがサポートされています。別のレプリカ数を入力すると、クラスタのプロビジョニングが失敗します。

制御プレーン/ワーカー仮想マシンのストレージ クラスが無効

次のコマンドを実行します。

```
kubectl describe ns <tkg-clsuter-namesapce>
```

TKG クラスタを作成する名前空間にストレージ クラスが割り当てられていることを確認します。そのストレージ クラスを参照する vSphere 名前空間 には ResourceQuota が必要で、そのストレージ クラスが スーパーバイザー 内にある必要があります。

スーパーバイザー にあるストレージ クラスと名前が一致することを確認します。vSphere 管理者として `kubectl get storageclasses` スーパーバイザー を実行します。WCP によって スーパーバイザー にストレージ プロファイルが適用されるときに、名前が変換されることがあります（ハイフンがアンダースコアになるなど）。

仮想マシン クラスが無効

クラスタ YAML で指定された値が、`kubectl get virtualmachineclass` によって返された仮想マシン クラスのいずれかと一致することを確認します。TKG クラスタではバインドされたクラスのみを使用できます。仮想マシン クラスは、vSphere 名前空間 に追加されるとバインドされます。

コマンド `kubectl get virtualmachineclasses` は スーパーバイザー 上のすべての仮想マシン クラスを返しますが、バインドされているクラスのみを使用できます。

TKR 分散が見つからない

次のようなエラーが表示される場合があります。

```
“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”
```

これは、コンテンツ ライブラリの問題である場合があります。使用可能になっているものを一覧表示するには、`kubectl get virtualmachineimages -A` コマンドを使用します。結果として、コンテンツ ライブラリで使用可能になっているもの、同期されたもの、またはアップロードされたものが表示されます。

スーパーバイザー 上の TKG には、新しい TKR API と互換性がある新しい TKR 名があります。コンテンツ ライブラリ内の各 TKR に正しい名前を付ける必要があります。

コンテンツ ライブラリでの名前：`photon-3-amd64-vmi-k8s-v1.23.8---vmware.2-tkg.1-zshippable`

TKG クラスタ仕様での対応する名前：`version: v1.23.8+vmware.2-tkg.1-zshippable`

TKG YAML は適用されるが、仮想マシンが作成されない

TKG 2.0 クラスタ YAML が有効で適用されていても、ノード仮想マシンが作成されない場合は、次のようにトラブルシューティングを実行します。

CAPI/CAPV リソースの確認

TKG によって CAPI/CAPV レベルのリソースが作成されたかどうかを確認します。

- CAPV によって VirtualMachine リソースが作成されたかどうかを確認します。
- 仮想マシン オペレータ ログを確認して、仮想マシンが作成されなかった理由を調べます。たとえば、ESX ホストでのリソース不足のために OVF のデプロイが失敗した可能性があります。
- CAPV と仮想マシン オペレータのログを確認します。
- NCP ログを確認します。NCP は制御プレーンの VirtualNetwork、VirtualNetworkInterface、および LoadBalancer を認識します。これらのリソースに関連するエラーがあると、問題になる可能性があります。

仮想マシン サービスのエラー

仮想マシン サービスのエラー

- 名前空間で `kubectl get virtualmachineservices` を実行します
- 仮想マシン サービスは作成されましたか。
- 名前空間で `kubectl describe virtualmachineservices` を実行します
- 仮想マシン サービスでエラーが報告されていますか。

仮想ネットワークのエラー

名前空間で `kubectl get virtualnetwork` を実行します。

このクラスタのために仮想ネットワークが作成されますか。

名前空間で `kubectl describe virtual network` を実行します。

仮想マシンのために仮想ネットワーク インターフェイスが作成されますか。

TKG クラスタ制御プレーンが実行されていない

TKG 制御プレーンが実行されていない場合は、エラーが発生したときにリソースの準備ができていたかどうかを確認します。起動していないのは Join Node 制御プレーンですか。それとも Init Node ですか。また、プロバイダ ID がノード オブジェクトに設定されていないかどうかも確認します。

エラーが発生したときにリソースの準備ができていたかどうかを確認する

ログを調べる他に、関連オブジェクトのステータス (ControlPlaneLoadBalancer) を確認すると、エラーが発生したときにリソースの準備ができていたかどうかを理解するのに役立ちます。ネットワークのトラブルシューティングを参照してください。

起動していないのは Join Node 制御プレーンですか。それとも Init Node ですか。

ノード参加が正しく動作しない場合があります。特定の仮想マシンのノード ログを調べてください。初期ノードが正常に起動しない場合、クラスタにワーカー ノードと制御プレーン ノードがない可能性があります。

プロバイダ ID がノード オブジェクトに設定されていない

仮想マシンが作成された場合は、IP を持っているかどうかを確認し、cloud-init ログを調べます（kubeadm コマンドが適切に実行されているか）。

CAPI コントローラのログを確認して、問題が発生しているかどうかを確認します。それは、TKG クラスタで `kubectl get nodes` を使用して確認できます。その後、ノード オブジェクトにプロバイダ ID があるかどうかを確認します。

TKG ワーカー ノードが作成されない

TKG クラスタと制御プレーン仮想マシンが作成されても、ワーカーが作成されないか、他の仮想マシン オブジェクトが作成されない場合は、次の手順を試します。

```
kubectl describe cluster CLUSTER-NAME
```

名前空間内の仮想マシン リソースを確認します。他に作成されたリソースはありますか。

ない場合は、CAPV ログを調べて、他の仮想マシン オブジェクトが作成されない理由を確認します。ブートストラップ データは使用できません。

CAPI がロード バランサを介して TKG クラスタ制御プレーン（ノード仮想マシンの IP アドレスを使用する NSX または外部ロード バランサを使用する Distributed Switch）と通信できない場合は、名前空間のシークレットを使用して TKG クラスタ kubeconfig を取得します。

名前空間のシークレットを使用して、TKG クラスタ kubeconfig を取得します。

```
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
```

これが「接続が拒否されました」で失敗する場合は、制御プレーンが適切に初期化されなかった可能性があります。I/O タイムアウトが発生する場合は、kubeconfig で IP アドレスへの接続を確認します。

組み込みのロード バランサを使用する NSX：

- 制御プレーン LB が起動し、アクセス可能であることを確認します。
- LB に IP がない場合は、NCP ログと NSX-T ユーザー インターフェイスを調べて、関連するコンポーネントが正しい状態であるかどうかを確認します。（NSX-T LB、VirtualServer、ServerPool は、すべて健全な状態である必要があります）。
- LB に IP アドレスがあっても、アクセスできない場合（`curl -k https://<LB- VIP>:6443/healthz` は不正なエラーを返します）。

  サービス外部 IP の LoadBalancer タイプが「保留中」状態の場合は、TKG クラスタがスーパーバイザー LB VIP を介してスーパーバイザー Kubernetes API と通信できることを確認します。IP アドレスの重複がないことを確認します。

TKG 制御プレーン ノードが健全な状態であるかどうかを確認します

- TKG クラスタ制御プレーンがエラー（プロバイダ ID でノードを作成できないなど）を報告しているかどうかを確認します。
- TKG クラスタ クラウド プロバイダがノードを正しいプロバイダ ID でマークしなかったため、CAPI はゲスト クラスタ ノードのプロバイダ ID とスーパーバイザー クラスタのマシン リソースを比較して検証することができません。

制御プレーン仮想マシンに SSH 接続するか、TKG クラスタ kubeconfig を使用して、TKG クラウド プロバイダ ポッドが実行中であることと、エラーがログに記録されているかどうかを確認します。Kubernetes 管理者およびシステム ユーザーとしての TKG サービス クラスタへの接続を参照してください。

```
kubectl get po -n vmware-system-cloud-provider
```

```
kubectl logs -n vmware-system-cloud-provider <pod name>
```

VMOP によって VirtualMachineService が正常に調整されなかった場合は、仮想マシン オペレータ ログを確認します。

NCP で NSX-T リソースを作成するときに問題が発生した場合は、NCP ログを確認します。

制御プレーンが適切に初期化されなかった場合は、仮想マシンの IP アドレスを特定します。ステータスには仮想マシンの IP が含まれている必要があります。

```
kubectl get virtualmachine -n <namespace> <TKC-name>-control-plane-0 -o yaml
```

```
ssh vmware-system-user@<vm-ip> -i tkc-cluster-ssh
```

kubeadm によってログにエラーが記録されているかどうかを確認します。

```
cat /var/log/cloud-init-output.log | less
```

プロビジョニングされた TKG クラスタが「作成中」フェーズで停止する

次のコマンドを実行して、クラスタのステータスを確認します。

```
kubectl get tkc -n <namespace>
```

```
kubectl get cluster -n <namespace>
```

```
kubectl get machines -n <namespace>
```

KubeadmConfig がありましたが、CAPI がそれを見つけることができませんでした。vmware-system-capv のトークンに kubeadmconfig をクエリする適切な権限があるかどうかを確認しました。

```
$kubectl --token=__TOKEN__ auth can-i get kubeadmconfig
yes
```

コントローラ ランタイム キャッシュが更新されていない可能性があります。CAPI ウォッチ キャッシュが古く、新しいオブジェクトを取得していない可能性があります。必要に応じて、問題を解決するために capi-controller-manager を再起動します。

```
kubectl rollout restart deployment capi-controller-manager -n vmware-system-capv
```

vSphere 名前空間 が「終了中」フェーズで停止する

バージョンの互換性の観点から、TKR、スーパーバイザー、および vCenter Server が同期していることを確認します。

名前空間は、名前空間の下のすべてのリソースが順番に削除された場合にのみ削除できます。

```
kubectl describe namespace NAME
```

次のエラーが見つかりました：「サーバからのエラー（Kubernetes ディストリビューションが見つかりません）：アドミッション Webhook「version.mutating.tanzukubernetescluster.run.tanzu.vmware.com」が要求を拒否しました：Kubernetes ディストリビューションが見つかりません」

vCenter Server の仮想マシン イメージを確認します。

```
kubectl get virtualmachineimages -A
```

TKG サービス クラスタ ノード エラーのトラブルシューティング

TKGS クラスタをプロビジョニング済みで、ノードが作成されていても、1 台以上の仮想マシンが起動しないかエラーになっている場合は、次のトラブルシューティングのヒントを試します。

関連する CRD の確認

仮想マシンが作成されたら、関連する CRD を作成する必要があります。関連する CRD が作成され、存在しているかどうかを確認します（machinedeployments、virtualmachines）。

マシンのデプロイを確認します。

```
kubectl get machinedeployments -A -o wide
```

仮想マシンを確認します。

```
kubectl get virtualmachines -A
```

ノード サイズの確認

TKG クラスタをプロビジョニングしました。システムは制御プレーンの仮想マシンのパワーオンを試行しますが、エラーが発生し、次のメッセージが表示されます。

```
The host does not have sufficient CPU resources to satisfy the reservation.
```

仮想マシンのサイズまたはクラスが、クラスタをデプロイするのに十分ではありません。仮想マシンのタイプまたはクラスを変更します。制御プレーンとワーカー ノードの両方に、極小規模および小規模の仮想マシン クラス タイプを使用しないでください。

TKG サービス クラスタ ネットワーク エラーのトラブルシューティング

TKGS クラスタ ネットワーク エラーのトラブルシューティングについては、このセクションのヒントを参照してください。

ノード ネットワークの確認

各 TKG クラスタには、次のネットワーク リソースが必要です。

ネットワーク オブジェクト	ネットワーク リソース	説明	トラブルシューティング	コマンド
VirtualNetwork	Tier-1 ルーターと、リンクされたセグメント	クラスタのノード ネットワーク	SNAT IP が割り当てられていることを確認します	`kubectl get virtualnetwork -n NS-NAME`
VirtualNetworkInterface	セグメントの論理ポート	クラスタ ノードのノード ネットワーク インターフェイス	各 VirtualMachine に IP アドレスがあることを確認します	`kubectl get virtualmachines -n NS-NAME NODE-NAME`

制御プレーンのロード バランサの確認

TKG クラスタ制御プレーンのロード バランサは、Kubernetes API サーバへのアクセスを提供します。このロード バランサは、クラスタの作成時にシステムによって自動的にプロビジョニングされます。次のリソースが必要です。

制御プレーン ロード バランサのステータスを調べると、エラーが発生したときにリソースの準備ができていたかどうかを理解するのに役立ちます。通常、このロード バランサを検索するには、スーパーバイザー クラスタに対して `kubectl get services -A | grep control-plane-service` というコマンドを使用します。

ネットワーク オブジェクト	ネットワーク リソース	説明	トラブルシューティング	コマンド
VirtualMachineService	該当なし	VirtualMachineService が作成され、k8s サービスに変換されます。	ステータスが更新され、ロード バランサの仮想 IP (VIP) が含まれていることを確認します。	`kubectl get virtualmachineservices -n NS-NAME SERVICE-NAME`

サービス	VirtualServer インスタンスおよび関連付けられているサーバ プール(メンバー プール)があるロード バランサ サーバ	TKG クラスタ API サーバにアクセスするために、ロード バランサ タイプの Kubernetes サービスが作成されます。	外部 IP アドレスが割り当てられていることを確認します。LB サービスの外部 IP を介して TKG クラスタ API にアクセスできることを確認します。	スーパーバイザー名前空間：
				`kubectl get services -A | grep control-plane-service`
				クラスタ名前空間：
				`kubectl get services -n NS-NAME`
				どちらかの名前空間
				`curl -k https://EXTERNAL-IP:PORT/healthz`

エンドポイント	エンドポイント メンバー(TKG クラスタ制御プレーン ノード) は、メンバー プールに含まれている必要があります。	すべての TKG クラスタ制御プレーン ノードを含むように、エンドポイントが作成されます。		`kubectl get endpoints -n NS-NAME SERVICE-NAME`

ワーカー ノード上のロード バランサ サービスの確認

LoadBalancer タイプの Kubernetes サービスが作成されるときに、TKG クラスタ ワーカー ノードのロード バランサ インスタンスがユーザーによって作成されます。

最初の手順は、クラウド プロバイダが TKG クラスタ上で実行されていることを確認することです。

```
kubectl get pods -n vmware-system-cloud-provider
```

関連する Kubernetes オブジェクトが作成され、正しい状態になっていることを確認します。

ネットワーク オブジェクト	ネットワーク リソース	説明	コマンド
スーパーバイザー上の VirtualMachineService	該当なし	VirtualMachineService がスーパーバイザーで作成され、スーパーバイザーで Kubernetes サービスに変換されます	`kubectl get virtualmachineservice -n NS-NAME SVC-NAME`
スーパーバイザー上のロード バランサ サービス	TKG クラスタ ロード バランサ内の VirtualServer、および関連付けられているメンバー プール。	ロード バランサ サービスは、このLB タイプのサービスにアクセスするために、スーパーバイザー上に作成されます	`kubectl get services -n NS-NAME SVC-NAME`
スーパーバイザー上のエンドポイント	エンドポイント メンバー（TKG クラスタ ワーカー ノード）は、NSX のメンバー プールに含まれている必要があります。	すべての TKG クラスタ ワーカー ノードを含むように、エンドポイントが作成されます	`# kubectl get endpoints -n NS-NAME SVC-NAME`
TKG クラスタのロード バランサ サービス	該当なし	ユーザーによってデプロイされた TKG クラスタのロード バランサ サービスは、ステータスがロード バランサの IP で更新される必要があります	`kubectl get services`

スーパーバイザー NSX ネットワーク スタックの確認

Kubernetes API サーバ、NCP ポッド、および任意のコントローラ ポッドで実行されるマネージャ コンテナは、インフラストラクチャ ネットワークの問題を確認するための主要な開始点になります。

次のエラー メッセージは、ESXi ホストの NIC が接続されている物理プロトコル グループなどのネットワーク ファブリックの任意のポイントで、ルーティングまたは MTU の問題が発生していることを示します。

```
{"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"}
```

トラブルシューティングを行うには、ESXi ホストに SSH 接続し、次のコマンドを実行します。

```
esxcli network ip interface ipv4 get
```

このコマンドは、ホストのすべての vmkernel インターフェイスを一覧表示します。TEP インターフェイスを 1 つ使用している場合、インターフェイスは常に vmk10 になります。2 番目または 3 番目の TEP インターフェイスを使用している場合、インターフェイスは vmk11 と vmk12 などになります。作成される TEP インターフェイスの数は、アップリンク プロファイルで TEP に割り当てたアップリンクの数によって異なります。アップリンク間の TEP に「ロード共有」を選択した場合は、アップリンクごとに TEP インターフェイスが 1 つ作成されます。

TEP 間で行われる主要な ping コマンドの構文は、次のとおりです。

```
vmkping ++netstack=vxlan -s 1572 -d -I vmk10 10.218.60.66
```

ここで、

- `-s` はパケット サイズです
- `-d` は断片化しないという意味です

- `-I` は、リンクの送信元が vmk10 であることを意味します
- `IP address` は、ping を送信している別の ESXi ホストまたは NSX Edge 上の TEP インターフェイスです

MTU が 1,600 に設定されている場合、1,573 を超えるパケット サイズは失敗します（1,500 を超える MTU がのみが必要）。MTU が 1,500 に設定されている場合、1,473 を超えるパケット サイズは失敗します。ping の送信元となる追加の TEP インターフェイスがある場合は、これを vmk11 に変更することができます。

失敗した TKG クラスタのアップグレードの再開

TKG クラスタの更新に失敗した場合は、更新ジョブを再開して更新をもう一度試すことができます。

問題

TKG クラスタの更新に失敗し、クラスタの状態が `upgradefailed` になります。

原因

クラスタ更新が失敗した場合、ストレージの不足などいくつかの理由が考えられます。失敗した更新ジョブを再開してクラスタ更新をもう一度試すには、次の手順を実行します。

解決方法

1 管理者として スーパーバイザー にログインします。

2 `update_job_name` を検索します。

```
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 `curl` を使用して要求を発行できるように、`kubectl proxy` を実行します。

```
kubectl proxy &
```

`Starting to serve on 127.0.0.1:8001` が表示されます。

注： `kubectl` を使用して、リソースの `.status` にパッチを適用したり、更新したりすることはできません。

4 `curl` を使用して次のパッチコマンドを発行し `.spec.backoffLimit` を発生させます。

```
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 ジョブ コントローラで新しいポッドが作成できるように、`curl` を使用して次のパッチコマンドを発行し `.status.conditions` をクリアします。

```
$ 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
```

コンテナ デプロイ エラーのトラブルシューティング

認証されたユーザーに対してポッド セキュリティ ポリシーおよびロールベースのアクセス コントロールが構成されていない場合は、コンテナ デプロイ エラーが発生することがあります。

問題

コンテナ ワークロードを TKG 2.0 クラスタにデプロイしても、ワークロードが起動しません。次のようなエラーが表示されます。

```
Error: container has runAsNonRoot and image will run as root.
```

原因

TKG クラスタは、PodSecurityPolicy アドミッション コントローラが有効になるようにプロビジョニングされています。認証されたユーザーは、クラスタ管理者が認証されたユーザーに PodSecurityPolicy をバインドするまで、権限のあるポッドまたは権限のないポッドを作成できません。

解決方法

TKR 1.24 以前を使用している場合は、デフォルトの PodSecurityPolicy に適したバインドを作成するか、カスタムの PodSecurityPolicy を定義します。TKR 1.25 以降を使用している場合は、ポッド セキュリティ アドミッションを構成します。18 章 TKG サービス クラスタのセキュリティの管理を参照してください。

コンテナ レジストリ エラーのトラブルシューティング

TKG クラスタ ポッドで外部コンテナ レジストリを使用できます。

コンテナ レジストリからイメージをプルするときに発生したエラーのトラブルシューティング

信頼する証明書を使用して TKG を構成し、自己署名証明書をクラスタの kubeconfig に追加すると、その自己署名証明書を使用するプライベート レジストリからコンテナ イメージを正常にプルできるようになります。

次のコマンドは、コンテナ イメージがポッド ワークロードに対して正常にプルされたかどうかを判断するのに役立ちます。

```
kubectl describe pod PODNAME
```

このコマンドを実行すると、特定のポッドの詳細なステータスとエラー メッセージが表示されます。カスタム証明書をクラスタに追加する前にイメージをプルする例を次に示します。

```
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
```

次のコマンドを実行します。

```
kubectl get pods
```

ポッド全体のステータス ビューに `ErrImagePull` エラーも表示されます。

```
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
```

エラー「x509: certificate signed by unknown authority」と「ErrImagePull」は、プライベート コンテナ レジストリに接続するための正しい証明書を使用してクラスタが構成されていないことを示しています。証明書が見つからないか、構成が正しくありません。

証明書の構成後にプライベート レジストリに接続するとエラーが発生する場合は、構成に適用された証明書がクラスタに適用されているかどうかを確認できます。構成に適用された証明書が正しく適用されているかどうかを確認するには、SSH を使用します。

SSH を使用してワーカー ノードに接続することで、2 つの調査手順を実行できます。

1 フォルダ `/etc/ssl/certs/` に `tkg-<cert_name>.pem` という名前のファイルがあるかを確認します。`<cert_name>` は、`TkgServiceConfiguration` に追加された証明書の「name」プロパティです。証明書が `TkgServiceConfiguration` 内の証明書と一致していてもプライベート レジストリの使用が機能しない場合は、次の手順を実行してさらに診断します。

2 `openssl s_client -connect hostname:port_num` コマンドを実行して、自己署名証明書を使用したターゲット サーバへの openssl 接続テストを実行します。`hostname` は自己署名証明書を使用しているプライベート レジストリのホスト名/DNS 名で、`port_num` はサービスが実行されているポート番号です（HTTPS の場合は通常 443）。

   自己署名証明書を使用しているエンドポイントに接続するときに openssl によって返されるエラーを正確に確認し、適切な証明書を `TkgServiceConfiguration` に追加するなどしてその状況を解決できます。TKG クラスタに誤った証明書が組み込まれている場合は、正しい証明書を使用して構成を更新し、TKG クラスタを削除してから、正しい証明書を含む構成を使用してクラスタを再作成する必要があります。

3 シークレットのデータ マップの内容が double 型の base64 エンコード形式であることを確認します。base64 二重エンコードは必須です。データ マップ値の内容が base6 二重エンコードでない場合は、結果の PEM ファイルを処理できません。

追加の信頼できる CA に関するエラーのトラブルシューティング

TKG クラスタに信頼できる CA 証明書を追加する際に問題が発生した場合は、このトピックを参照してください。

追加の信頼できる CA に関するエラーのトラブルシューティング

v1alpha3 API または v1beta1 API のいずれかを使用して、追加の信頼できる CA 証明書の値を含む `trust` 変数を指定できます。一般的な使用事例は、プライベート コンテナ レジストリ証明書をクラスタに追加する場合です。TKG サービス クラスタとプライベート コンテナ レジストリの統合を参照してください。

v1beta1 API を使用する場合の例は次のとおりです。

```
topology:
  variables:
    - name: trust
      value:
        additionalTrustedCAs:
          - name: my-ca
```

シークレットは次のようになります。

```
apiVersion: v1
data:
  my-ca: # Double Base64 encoded CA certificate
kind: Secret
metadata:
  name: CLUSTER_NAME-user-trusted-ca-secret
  namespace: tap
type: Opaque
```

TKG クラスタ仕様に `trust.additionalTrustedCAs` スタンザを追加すると、スーパーバイザー は TKG サービス クラスタのローリング アップデート モデルについて方式でクラスタ ノードを更新します。ただし、`trust` 値が誤っていると、マシンが適切に起動せず、クラスタへの参加に失敗します。

v1beta1 API を使用している場合は、証明書の内容を double 型の base64 エンコード形式にする必要があります。証明書の内容が double 型の base64 エンコード形式でない場合は、次のエラーが表示されることがあります。

```
ls cannot access '/var/tmp/_var_ib_containerd': No such file or directory
```

v1alpha3 API（または v1alpha2 API）を使用している場合は、証明書の内容を single 型の base64 エンコード形式にする必要があります。証明書の内容が base64 エンコード形式でない場合は、次のエラーが表示されることがあります。

```
"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
```

適切なエンコードを使用しないと、マシン ノードが起動せず、上記のエラーが表示されます。この問題を解決するには、証明書の内容を適切にエンコードして、その値をシークレットのデータ マップに追加します。

「kubectl replace -f /tmp/kubectl-edit-2005376329.yaml」を実行して、この更新を再試行できます。