Handbuch zu VMware Private AI Foundation with NVIDIA

23. Juli 2024

VMware Cloud Foundation 5.2

vmware by Broadcom

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

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

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

Inhalt

Informationen zum VMware Private AI Foundation with NVIDIA-Handbuch

Das *VMware Private AI Foundation with NVIDIA-Handbuch* bietet einen Überblick über die Komponenten von VMware Private AI Foundation with NVIDIA und allgemeine Workflows für Entwicklungs- und Produktionsanwendungsfälle.

Zielgruppe

Die Informationen im *VMware Private AI Foundation with NVIDIA-Handbuch* wurden für erfahrene Datencenter-Cloudadministratoren, Datenwissenschaftler und DevOps-Ingenieure verfasst, die mit Folgendem vertraut sind:

- Cloud-Administratoren
  - Virtualisierungskonzepten und Software-Defined Data Center (SDDCs)
  - Hardwarekomponenten wie Top-of-Rack (ToR)-Switches, Rack-übergreifende Switches, Server mit direkt angeschlossenem Speicher, Kabel und Netzteile
  - Methoden zur Einrichtung von NVIDIA GPUs auf Servern in einem Datencenter
  - Verwenden von VMware vSphere® für die Arbeit mit virtuellen Maschinen.
  - Verwenden von vSphere IaaS control plane zum Konfigurieren und Zuweisen von vSphere-Ressourcen zu vSphere-Namespaces auf einem Supervisor.

  Zeigen Sie als Cloud-Administrator folgende Informationen an:

  - Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten
  - Kapitel 3 Bereitstellen einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA
  - Kapitel 6 Überwachen von VMware Private AI Foundation with NVIDIA
- Datenwissenschaftler
  - Container, einschließlich Docker, Helm-Diagramme und Harbor-Registrierung

  Zeigen Sie als Datenwissenschaftler folgende Informationen an:

  - Kapitel 3 Bereitstellen einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA

  - Kapitel 5 Bereitstellen von RAG-Arbeitslasten in VMware Private AI Foundation with NVIDIA
- DevOps-Ingenieure
  - Bereitstellen virtueller Maschinen in vSphere mithilfe der Kubernetes-API.
  - Container, einschließlich Docker, Helm-Diagramme und Harbor-Registrierung
  - Arbeiten mit vSphere IaaS control plane zur Bereitstellung von VMs und Tanzu Kubernetes Grid-Clustern (TKG).

  Zeigen Sie als DevOps-Ingenieur folgende Informationen an:

  - Kapitel 4 Bereitstellen von KI-Arbeitslasten auf TKG-Clustern in VMware Private AI Foundation with NVIDIA
  - Kapitel 5 Bereitstellen von RAG-Arbeitslasten in VMware Private AI Foundation with NVIDIA

VMware-Softwarekomponenten

Die Funktionen der VMware Private AI Foundation with NVIDIA-Lösung sind abhängig von Ihrer Rolle in der Organisation über mehrere Softwarekomponenten hinweg verfügbar.

Rolle „Zielbenutzer“	Softwarekategorie	Unterstützte Softwareversionen
Cloud-Administratoren	Komponenten, die in VMware Cloud Foundation bereitgestellt werden	Weitere Informationen finden Sie unter VMware-Komponenten in VMware Private AI Foundation with NVIDIA.
Datenwissenschaftler	Deep Learning-VM-Komponenten	Weitere Informationen finden Sie unter Versionshinweise zu VMware Deep Learning VM.
DevOps-Ingenieure	TK-Versionen (TKr)	Weitere Informationen finden Sie unter Versionshinweise zu VMware Tanzu Kubernetes-Versionen.

Zugehörige VMware-Dokumentation

Die VMware Private AI Foundation with NVIDIA-Lösung enthält ein Paket von VMware-Softwareprodukten und -komponenten. Die Dokumentation für diese Softwareprodukte lautet wie folgt:

- VMware Cloud Foundation-Dokumentation
- Dokumentation zu VMware vSphere und vSAN
- Dokumentation zur VMware vSphere IaaS Control Plane
- Dokumentation zu VMware Aria Automation
- Dokumentation zu VMware Aria Operations
- Dokumentation zu VMware Aria Suite Lifecycle

- Dokumentation zu VMware Data Services Manager

VMware Cloud Foundation-Glossar

Im VMware Cloud Foundation-Glossar sind Begriffe definiert, die speziell für VMware Cloud Foundation gelten.

Was ist VMware Private AI Foundation with NVIDIA?

1

Als Lösung mit mehreren Komponenten können Sie VMware Private AI Foundation with NVIDIA verwenden, um generative KI-Arbeitslasten auszuführen, indem Sie beschleunigtes Computing von NVIDIA sowie die Verwaltung der virtuellen Infrastruktur und Cloud-Verwaltung von VMware Cloud Foundation verwenden.

VMware Private AI Foundation with NVIDIA bietet eine Plattform für die Bereitstellung von KI-Arbeitslasten auf ESXi-Hosts mit NVIDIA GPUs. Darüber hinaus wird die Ausführung von KI-Arbeitslasten basierend auf NVIDIA GPU Cloud (NGC)-Containern speziell durch VMware validiert.

VMware Private AI Foundation with NVIDIA unterstützt zwei Anwendungsfälle:

Anwendungsfall Entwicklung

Cloud-Administratoren und DevOps-Ingenieure können KI-Arbeitslasten, einschließlich Retrieval-Augmented Generation (RAG), in Form von Deep Learning-VMs bereitstellen. Datenwissenschaftler können diese Deep Learning-VMs für die KI-Entwicklung verwenden. Weitere Informationen finden Sie unter Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA.

Anwendungsfall Produktion

Cloud-Administratoren können DevOps-Ingenieuren eine VMware Private AI Foundation with NVIDIA-Umgebung für die Bereitstellung produktionsbereiter KI-Arbeitslasten auf Tanzu Kubernetes Grid-Clustern (TKG) in vSphere IaaS control plane bereitstellen.

Informationen zu den Komponenten, die Teil der VMware Private AI Foundation with NVIDIA Lösung sind, und deren Architektur in VMware Cloud Foundation finden Sie unter Systemarchitektur von VMware Private AI Foundation with NVIDIA.

Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten

2

Als Cloud-Administrator müssen Sie spezifische Software bereitstellen und die VI-Arbeitslastdomänen des Ziels so konfigurieren, dass Datenwissenschaftler und DevOps-Ingenieure KI-Arbeitslasten zusätzlich zu VMware Private AI Foundation with NVIDIA bereitstellen können.

VMware-Komponenten in VMware Private AI Foundation with NVIDIA

Die Funktionen der VMware Private AI Foundation with NVIDIA-Lösung sind über mehrere Softwarekomponenten hinweg verfügbar.

- VMware Cloud Foundation 5.2
- VMware Aria Automation 8.18 und VMware Aria Automation 8.18
- VMware Aria Operations 8.18 und VMware Aria Operations 8.18
- VMware Data Services Manager 2.1

Informationen zur Architektur und zu den Komponenten von VMware Private AI Foundation with NVIDIA finden Sie unter Systemarchitektur von VMware Private AI Foundation with NVIDIA.

Bereitstellungsworkflow für VMware Private AI Foundation with NVIDIA

Die Funktionen von VMware Private AI Foundation with NVIDIA basieren auf einem grundlegenden Komponentensatz sowie zusätzlichen Komponenten, die für die Bereitstellung einer der folgenden KI-Arbeitslasttypen erforderlich sind:

- Deep Learning-VMs im Allgemeinen
- KI-Arbeitslasten in einem GPU-beschleunigten TKG-Cluster im Allgemeinen
- RAG-Arbeitslasten als Deep Learning-VMs oder Anwendungen auf GPU-beschleunigten TKG-Clustern

Mit der Bereitstellung einer RAG-Arbeitslast wird der allgemeine Ansatz für Deep Learning-VMs und KI-Arbeitslasten auf TKG-Clustern um die Bereitstellung einer PostgreSQL-Datenbank vom Typ „pgvector“ und die Konfiguration der Anwendung mit der pgvector-Datenbank erweitert.

In einer nicht verbundenen Umgebung müssen Sie zusätzliche Schritte zum Einrichten und Bereitstellen von Appliances durchführen und Ressourcen lokal bereitstellen, damit Ihre Arbeitslasten darauf zugreifen können.

Verbundene Umgebung

Aufgabe	Anwendungsbeispiele für die Bereitstellung von KI-Arbeitslasten	Schritte
Überprüfen Sie die Architektur und die Anforderungen für die Bereitstellung von VMware Private AI Foundation with NVIDIA.	Alle	■ Systemarchitektur von VMware Private AI Foundation with NVIDIA ■ Anforderungen für die Bereitstellung von VMware Private AI Foundation with NVIDIA
Konfigurieren Sie eine Lizenzdienst-Instanz im NVIDIA-Lizenzierungsportal und generieren Sie ein Clientkonfigurationstoken.		Benutzerhandbuch für das NVIDIA-Lizenzsystem.
Generieren Sie einen API-Schlüssel für den Zugriff auf den NVIDIA NGC-Katalog.		Abrufen und Ausführen von NVIDIA AI Enterprise-Containern
Erstellen Sie eine Inhaltsbibliothek für Deep Learning-VM-Images.	Bereitstellen einer Deep Learning-VM	Erstellen einer Inhaltsbibliothek mit Deep Learning-VM-Images für VMware Private AI Foundation with NVIDIA
Aktivieren Sie vSphere IaaS control plane (wurde früher als vSphere with Tanzu bezeichnet).	Alle	Konfigurieren von vSphere IaaS Control Plane für VMware Private AI Foundation with NVIDIA
Bereitstellen Stellen Sie VMware Aria Automation mithilfe von VMware Aria Suite Lifecycle in VMware Cloud Foundation mode bereit.	Alle Erforderlich, wenn Datenwissenschaftler und DevOps-Ingenieure Arbeitslasten mithilfe von Self-Service-Katalogelementen in VMware Aria Automation bereitstellen.	1 Private Cloud-Automatisierung für VMware Cloud Foundation 2 Einrichten von VMware Aria Automation für VMware Private AI Foundation with NVIDIA
Stellen Sie VMware Aria Operations mithilfe von VMware Aria Suite Lifecycle in VMware Cloud Foundation mode bereit.	Alle	Intelligentes Betriebsmanagement für VMware Cloud Foundation.

Aufgabe	Anwendungsbeispiele für die Bereitstellung von KI-Arbeitslasten	Schritte
VMware Data Services Manager bereitstellen	Bereitstellen einer RAG-Arbeitslast	1 Installieren und Konfigurieren von VMware Data Services Manager Sie stellen eine VMware Data Services Manager-Instanz in der Verwaltungsdomäne bereit. 2 Erstellen eines Vektordatenbank-Katalogelements in VMware Aria Automation
Richten Sie eine Maschine mit Zugriff auf die Supervisor-Instanz ein, die über Docker- und Helm-Tools sowie Kubernetes CLI Tools for vSphere verfügt.	Alle Erforderlich, wenn die KI-Arbeitslasten durch direkte Verwendung des Befehls `kubectl` bereitgestellt werden.	Installieren des Kubernetes CLI Tools for vSphere

Getrennte Umgebung

Aufgabe	Bereitstellungsoptionen für zugehörige KI-Arbeitslasten	Schritte
Überprüfen Sie die Anforderungen für die Bereitstellung von VMware Private AI Foundation with NVIDIA.	Alle	■ Systemarchitektur von VMware Private AI Foundation with NVIDIA ■ Anforderungen für die Bereitstellung von VMware Private AI Foundation with NVIDIA
Stellen Sie eine Dienstinstanz für delegierte NVIDIA-Lizenzen bereit.		Installieren und Konfigurieren der virtuellen DLS-Appliance Sie können die virtuelle Appliance in derselben Arbeitslastdomäne wie die KI-Arbeitslasten oder in der Verwaltungsdomäne bereitstellen.
1 Registrieren Sie eine NVIDIA DLS Instanz auf dem NVIDIA Lizenzierungsportal, binden Sie einen Lizenzserver daran und installieren Sie ihn. 2 Generieren Sie einen Clientauthentifizierungstoken.		■ Konfigurieren einer Dienstinstanz ■ Verwalten von Lizenzen auf einem Lizenzserver.
Erstellen Sie eine Inhaltsbibliothek für Deep Learning-VM-Images	Bereitstellen einer Deep Learning-VM	Erstellen einer Inhaltsbibliothek mit Deep Learning-VM-Images für VMware Private AI Foundation with NVIDIA
Aktivieren Sie vSphere IaaS control plane (wurde früher als vSphere with Tanzu bezeichnet)	Alle	Konfigurieren von vSphere IaaS Control Plane für VMware Private AI Foundation with NVIDIA

Aufgabe	Bereitstellungsoptionen für zugehörige KI-Arbeitslasten	Schritte
■ Richten Sie eine Maschine ein, die Zugriff auf das Internet hat und auf der Docker und Helm installiert sind. ■ Richten Sie eine Maschine ein, die Zugriff auf vCenter Server für die VI-Arbeitslastdomäne, die Supervisor-Instanz und die lokale Containerregistrierung hat. Die Maschine muss über Docker-, Helm- und Kubernetes CLI Tools for vSphere verfügen.		■ Bereitstellen eines Bastion-Hosts ■ Installieren des Kubernetes CLI Tools for vSphere
Konfigurieren Sie eine Inhaltsbibliothek für Tanzu Kubernetes Releases (TKr) für Ubuntu	■ Bereitstellen einer RAG-Arbeitslast in einem GPU-beschleunigten TKG-Cluster ■ Bereitstellen von KI-Arbeitslasten in einem GPU-beschleunigten TKG-Cluster	Konfigurieren einer Inhaltsbibliothek mit Ubuntu TKr für eine getrennte VMware Private AI Foundation with NVIDIA-Umgebung
Richten Sie einen Harbor-Registrierungsdienst im Supervisor ein.	Alle Erforderlich, wenn die KI-Arbeitslasten auf einem Supervisor in vSphere IaaS control plane bereitgestellt werden In einer Umgebung ohne vSphere IaaS control plane müssen Sie zum Abrufen von Container-Images auf einer Deep Learning-VM, die direkt auf einem vSphere-Cluster ausgeführt wird, eine Registrierung eines anderen Anbieters konfigurieren.	Einrichten einer Private Harbor-Registrierung in VMware Private AI Foundation with NVIDIA
Laden Sie die Komponenten der NVIDIA-Operatoren in die Umgebung hoch.	■ Bereitstellen einer RAG-Arbeitslast in einem GPU-beschleunigten TKG-Cluster ■ Bereitstellen von KI-Arbeitslasten in einem GPU-beschleunigten TKG-Cluster	Hochladen der NVIDIA GPU-Operatorkomponenten in eine getrennte Umgebung

Aufgabe	Bereitstellungsoptionen für zugehörige KI-Arbeitslasten	Schritte
Geben Sie einen Speicherort zum Herunterladen der vGPU-Gasttreiber an.	Bereitstellen einer Deep Learning-VM	Laden Sie die erforderlichen aus dem NVIDIA-Lizenzierungsportal heruntergeladenen vGPU-Gasttreiberversionen und einen Index in einem der folgenden Formate auf einen lokalen Webserver hoch: ■ Eine Indexdatei vom Typ `.txt` mit einer Liste der Dateien vom Typ `.run` oder `.zip` der vGPU-Gasttreiber. *host-driver-version-1* *guest-driver-download-URL-1* *host-driver-version-2* *guest-driver-download-URL-2* *host-driver-version-3* *guest-driver-download-URL-3* ■ Ein Verzeichnisindex im Format, das von Webservern wie NGINX und Apache HTTP Server generiert wird. Die versionsspezifischen vGPU-Treiberdateien müssen als Dateien vom Typ `.zip` bereitgestellt werden.
Laden Sie die NVIDIA NGC-Container-Images in eine private Containerregistrierung hoch, z. B. in den Harbor-Registrierungsdienst des Supervisors.	Alle In einer Umgebung ohne vSphere IaaS control plane müssen Sie zum Abrufen von Container-Images auf einer Deep Learning-VM, die direkt auf einem vSphere-Cluster ausgeführt wird, eine Registrierung eines anderen Anbieters konfigurieren.	Hochladen von KI-Container-Images in eine Private Harbor-Registrierung in VMware Private AI Foundation with NVIDIA
Stellen Sie VMware Aria Automation mithilfe von VMware Aria Suite Lifecycle in VMware Cloud Foundation mode bereit.	Alle Erforderlich, wenn Datenwissenschaftler und DevOps-Ingenieure Arbeitslasten mithilfe von Self-Service-Katalogelementen in VMware Aria Automation bereitstellen.	1 Private Cloud-Automatisierung für VMware Cloud Foundation 2 Einrichten von VMware Aria Automation für VMware Private AI Foundation with NVIDIA

Aufgabe	Bereitstellungsoptionen für zugehörige KI-Arbeitslasten	Schritte
Stellen Sie VMware Aria Operations mithilfe von VMware Aria Suite Lifecycle in VMware Cloud Foundation mode bereit.	Alle	Intelligentes Betriebsmanagement für VMware Cloud Foundation
VMware Data Services Manager bereitstellen	Bereitstellen einer RAG-Arbeitslast	1 Installieren und Konfigurieren von VMware Data Services Manager Sie stellen eine VMware Data Services Manager-Instanz in der Verwaltungsdomäne bereit. 2 Erstellen eines Vektordatenbank-Katalogelements in VMware Aria Automation

Lesen Sie als Nächstes die folgenden Themen:

- Systemarchitektur von VMware Private AI Foundation with NVIDIA
- Anforderungen für die Bereitstellung von VMware Private AI Foundation with NVIDIA
- Erstellen einer Inhaltsbibliothek mit Deep Learning-VM-Images für VMware Private AI Foundation with NVIDIA
- Konfigurieren von vSphere IaaS Control Plane für VMware Private AI Foundation with NVIDIA
- Konfigurieren einer Inhaltsbibliothek mit Ubuntu TKr für eine getrennte VMware Private AI Foundation with NVIDIA-Umgebung
- Einrichten einer Private Harbor-Registrierung in VMware Private AI Foundation with NVIDIA
- Hochladen der NVIDIA GPU-Operatorkomponenten in eine getrennte Umgebung
- Einrichten von VMware Aria Automation für VMware Private AI Foundation with NVIDIA

Systemarchitektur von VMware Private AI Foundation with NVIDIA

VMware Private AI Foundation with NVIDIA wird zusätzlich zu VMware Cloud Foundation ausgeführt und bietet Unterstützung für KI-Arbeitslasten in VI-Arbeitslastdomänen mit vSphere IaaS control plane, das mithilfe von kubectl und VMware Aria Automation bereitgestellt wird.

Abbildung 2-1. Beispielarchitektur für VMware Private AI Foundation with NVIDIA

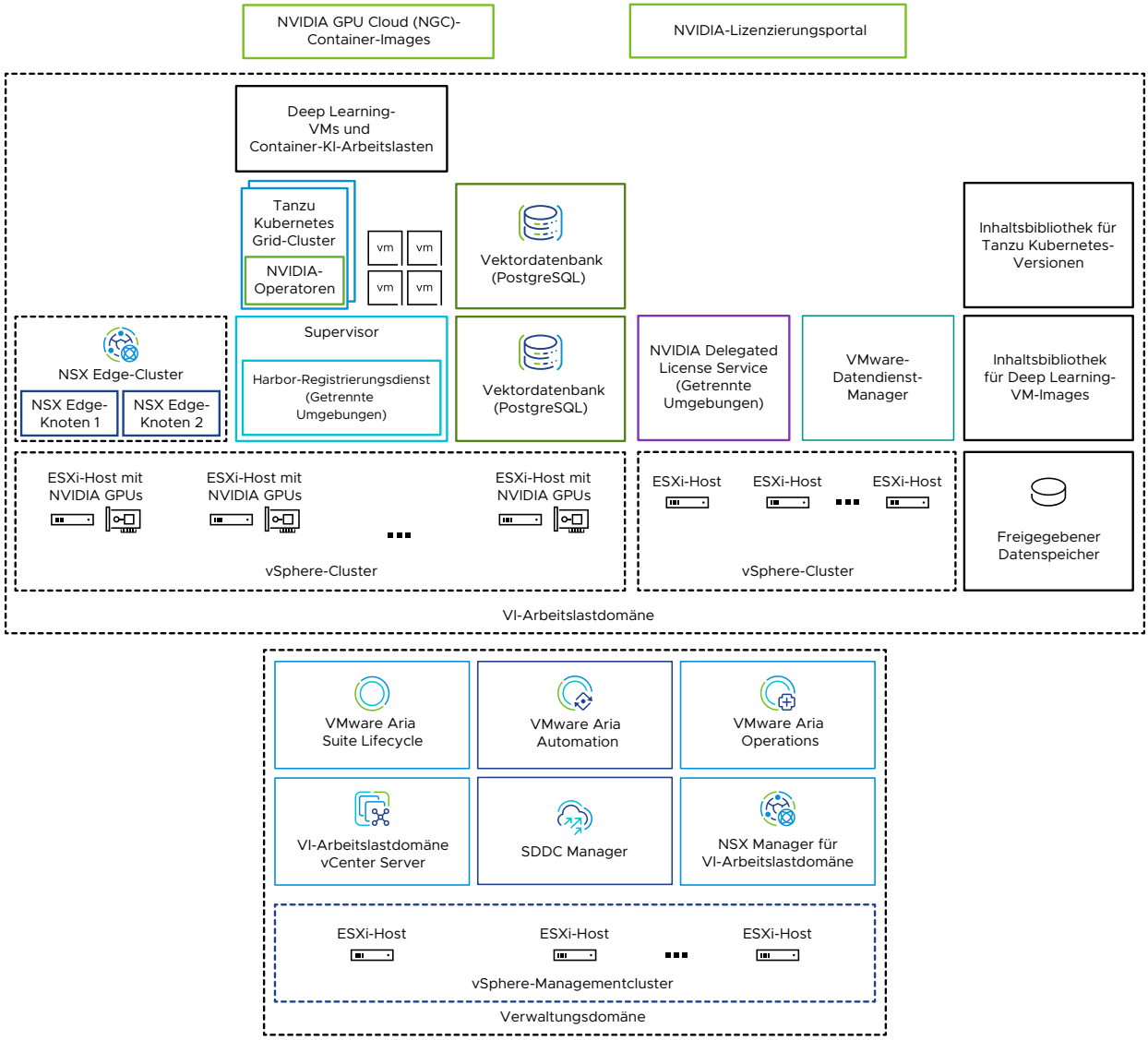


Tabelle 2-1. Komponenten zum Ausführen von KI-Arbeitslasten in VMware Private AI Foundation with NVIDIA

Komponente	Beschreibung
GPU-aktivierte ESXi-Hosts	ESXi Hosts, die wie folgt konfiguriert wurden: ■ Sie verfügen über eine NVIDIA-GPU, die für VMware Private AI Foundation with NVIDIA unterstützt wird. Die GPU wird von den Arbeitslasten gemeinsam genutzt, indem der Mechanismus der Zeitaufteilung oder der Mehrfachinstanz-GPU (MIG) verwendet wird. ■ Installieren Sie den NVIDIA vGPU-Hostmanagertreiber, sodass Sie vGPU-Profile basierend auf MIG oder der Zeitaufteilung verwenden können.
Supervisor	Mindestens ein vSphere-Cluster, der für vSphere IaaS control plane aktiviert ist, sodass Sie virtuelle Maschinen und Container auf vSphere mithilfe der Kubernetes-API ausführen können. Ein Supervisor ist selbst ein Kubernetes-Cluster, der als Steuerungsebene zum Verwalten von Arbeitslastclustern und virtuellen Maschinen dient.
Harbor-Registrierung	Eine lokale Image-Registrierung in einer getrennten Umgebung, in der Sie die aus dem NVIDIA NGC-Katalog heruntergeladenen Container-Images hosten.
NSX Edge-Cluster	Ein Cluster aus NSX Edge-Knoten, der zweistufiges Nord-Süd-Routing für den Supervisor und die ausgeführten Arbeitslasten bereitstellt. Das Tier-0-Gateway auf dem NSX Edge-Cluster befindet sich im Aktiv/Aktiv-Modus.
NVIDIA-Operatoren	■ NVIDIA GPU-Operator. Automatisiert die Verwaltung aller NVIDIA-Softwarekomponenten, die für die Bereitstellung von GPU für Container in einem Kubernetes-Cluster erforderlich sind. Der NVIDIA GPU-Operator wird auf einem TKG-Cluster bereitgestellt. ■ NVIDIA-Netzwerkoperator. Der NVIDIA-Netzwerkoperator hilft auch bei der Konfiguration der richtigen mellanox-Treiber für Container mithilfe virtueller Funktionen für Hochgeschwindigkeitsnetzwerke, RDMA und GPUDirect. Der Netzwerkoperator arbeitet mit dem GPU-Operator zusammen, um GPUDirect RDMA auf kompatiblen Systemen zu aktivieren. Der NVIDIA-Netzwerkoperator wird auf einem TKG-Cluster bereitgestellt.

Tabelle 2-1. Komponenten zum Ausführen von KI-Arbeitslasten in VMware Private AI Foundation with NVIDIA (Fortsetzung)

Komponente	Beschreibung
Vektordatenbank	Eine PostgreSQL-Datenbank, bei der die pgvector-Erweiterung aktiviert ist, sodass Sie sie in RAG-KI-Arbeitslasten (Retrieval Augmented Generation) verwenden können.
■ NVIDIA-Lizenzierungsportal ■ NVIDIA Delegated License Service (DLS)	Sie verwenden das NVIDIA-Lizenzierungsportal, um ein Clientkonfigurationstoken zu generieren, mit dem Sie dem vGPU-Gasttreiber in der Deep Learning-VM und den GPU-Operatoren auf TKG-Clustern eine Lizenz zuweisen. In einer getrennten Umgebung oder damit Ihre Arbeitslasten Lizenzinformationen erhalten, ohne eine Internetverbindung zu verwenden, hosten Sie die NVIDIA-Lizenzen lokal auf einer DLS-Appliance (Delegated License Service).
Inhaltsbibliothek	Inhaltsbibliotheken speichern die Images für die Deep Learning-VMs und für die Tanzu Kubernetes-Versionen. Sie verwenden diese Images für die Bereitstellung von KI-Arbeitslasten innerhalb der VMware Private AI Foundation with NVIDIA-Umgebung. In einer verbundenen Umgebung beziehen die Inhaltsbibliotheken ihre Inhalte aus den von VMware verwalteten öffentlichen Inhaltsbibliotheken. In einer nicht verbundenen Umgebung müssen Sie die erforderlichen Images manuell hochladen oder von einem internen Spiegelserver der Inhaltsbibliothek abrufen.
NVIDIA GPU Cloud (NGC)-Katalog	Ein Portal für GPU-optimierte Container für KI und maschinelles Lernen, die getestet wurden und für die lokale Ausführung auf unterstützten NVIDIA-GPUs zusätzlich zu VMware Private AI Foundation with NVIDIA bereit sind.

Als Cloud-Administrator verwenden Sie die Verwaltungskomponenten in VMware Cloud Foundation

Tabelle 2-2. Verwaltungskomponenten in VMware Private AI Foundation with NVIDIA

Verwaltungskomponente	Beschreibung
SDDC Manager	Sie verwenden SDDC Manager für die folgenden Aufgaben: ■ Bereitstellen einer GPU-fähigen VI-Arbeitslastdomäne, die auf vSphere Lifecycle Manager-Images basiert, und Hinzufügen von Clustern. ■ Bereitstellen eines NSX Edge-Clusters in VI-Arbeitslastdomänen für die Verwendung durch Supervisor-Instanzen und in der Verwaltungsdomäne für die VMware Aria Suite-Komponenten von VMware Private AI Foundation with NVIDIA. ■ Bereitstellen einer VMware Aria Suite Lifecycle-Instanz, die in das SDDC Manager-Repository integriert ist.
VI-Arbeitslastdomäne-vCenter Server	Sie verwenden diese vCenter Server-Instanz, um einen Supervisor zu aktivieren und zu konfigurieren.
NSX Manager für VI-Arbeitslastdomäne	SDDC Manager verwendet diese NSX Manager-Instanz, um NSX Edge-Cluster bereitzustellen und zu aktualisieren.
VMware Aria Suite Lifecycle	Sie verwenden VMware Aria Suite Lifecycle, um VMware Aria Automation und VMware Aria Operations bereitzustellen und zu aktualisieren.
VMware Aria Automation	Sie verwenden VMware Aria Automation, um Self-Service-Katalogelemente für die Bereitstellung von KI-Arbeitslasten für DevOps-Ingenieure und Datenwissenschaftler hinzuzufügen.
VMware Aria Operations	Sie verwenden VMware Aria Operations für die Überwachung der GPU-Auslastung in den GPU-fähigen Arbeitslastdomänen.
VMware Data Services Manager	Sie verwenden VMware Data Services Manager, um Vektordatenbanken zu erstellen, z. B. eine PostgreSQL-Datenbank mit der pgvector-Erweiterung.

Anforderungen für die Bereitstellung von VMware Private AI Foundation with NVIDIA

Sie stellen Komponenten von VMware Private AI Foundation with NVIDIA in Ihrer VMware Cloud Foundation-Umgebung in einer VI-Arbeitslastdomäne bereit, in der bestimmte NVIDIA-Komponenten installiert sein müssen.

Erforderliche VMware-Softwareversionen

Weitere Informationen finden Sie unter VMware-Komponenten in VMware Private AI Foundation with NVIDIA.

Unterstützte NVIDIA GPU-Geräte

Bevor Sie mit der Verwendung von VMware Private AI Foundation with NVIDIA beginnen, stellen Sie sicher, dass die GPUs auf Ihren ESXi-Hosts von VMware von Broadcom unterstützt werden:

Tabelle 2-3. Unterstützte NVIDIA-Komponenten für VMware Private AI Foundation with NVIDIA

NVIDIA-Komponente	Unterstützte Optionen
NVIDIA GPUs	■ NVIDIA A100 ■ NVIDIA L40S ■ NVIDIA H100
GPU-Freigabemodus	■ Zeitaufteilung ■ GPU mit mehreren Instanzen (MIG)

Erforderliche NVIDIA-Software

Das NVIDIA GPU-Gerät muss die aktuellen vGPU-Profile von NVIDIA AI Enterprise (NVAIE) unterstützen. Weitere Informationen finden Sie in der Dokumentation Von der NVIDIA Virtual GPU-Software unterstützte GPUs.

- NVIDIA vGPU-Hosttreiber (einschließlich VIB für ESXi-Hosts), der mit Ihrer VMware Cloud Foundation-Version kompatibel ist. Weitere Informationen finden Sie unter Versionshinweise zur virtuellen GPU-Software für VMware vSphere.
- NVIDIA GPU-Operator, der mit der Kubernetes-Version der bereitgestellten TKG-Cluster kompatibel ist. Weitere Informationen finden Sie unter Versionshinweise zum NVIDIA GPU-Operator und Versionshinweise zu VMware Tanzu Kubernetes-Versionen.

Erforderliches VMware Cloud Foundation-Setup

Bevor Sie VMware Private AI Foundation with NVIDIA bereitstellen, muss eine bestimmte Konfiguration in VMware Cloud Foundation verfügbar sein.

- Eine VMware Cloud Foundation-Lizenz.
- Eine VMware Private AI Foundation with NVIDIA-Add-On-Lizenz.

  Sie benötigen die VMware Private AI Foundation with NVIDIA-Add-On-Lizenz, um auf die folgenden Funktionen zuzugreifen:

  - Private AI-Einrichtung in VMware Aria Automation für Katalogelemente zwecks einfacher Bereitstellung von GPU-beschleunigten Deep Learning-VMs und TKG-Clustern.
  - Bereitstellung von PostgreSQL-Datenbanken mit der pgvector-Erweiterung mit Enterprise-Unterstützung.

- Bereitstellen und Verwenden des Deep Learning-VM-Images, das von VMware von Broadcom bereitgestellt wird.

Sie können KI-Arbeitslasten mit und ohne aktivierten Supervisor bereitstellen und die GPU-Metriken in vCenter Server und VMware Aria Operations unter der VMware Cloud Foundation-Lizenz verwenden.

- Lizenziertes NVIDIA vGPU-Produkt, einschließlich der VIB-Datei des Hosttreibers für ESXi-Hosts und der Treiber des Gastbetriebssystems. Weitere Informationen finden Sie in der Dokumentation Von der NVIDIA Virtual GPU-Software unterstützte GPUs.
- Die VIB-Datei des NVIDIA vGPU-Hosttreibers, die von https://nvid.nvidia.com/ heruntergeladen wurde
- Ein vSphere Lifecycle Manager-Image mit der VIB-Datei des in SDDC Manager verfügbaren vGPU-Hostmanagertreibers. Weitere Informationen finden Sie unter Verwalten von vSphere Lifecycle Manager-Images in VMware Cloud Foundation.
- Eine VI-Arbeitslastdomäne mit mindestens 3 ESXi GPU-fähigen Hosts, die auf dem vSphere Lifecycle Manager-Image basiert, das die VIB-Datei des Hostmanagertreibers enthält. Weitere Informationen finden Sie unter Bereitstellen einer VI-Arbeitslastdomäne mithilfe der SDDC Manager-Benutzeroberfläche und Verwalten von vSphere Lifecycle Manager-Images in VMware Cloud Foundation.
- Der NVIDIA vGPU-Hosttreiber ist auf jedem ESXi-Host im Cluster für KI-Arbeitslasten installiert und für vGPU konfiguriert.

  a Aktivieren Sie auf jedem ESXi-Host SR-IOV im BIOS und auf den Grafikgeräten für KI-Vorgänge direkt freigegeben.

  Informationen zum Konfigurieren von SR-IOV finden Sie in der Dokumentation Ihres Hardwareanbieters. Informationen zum Konfigurieren der Option „Direkt freigegeben“ auf Grafikgeräten finden Sie unter Konfigurieren von virtuellen Grafiken in vSphere.

  b Installieren Sie den NVIDIA vGPU-Hostmanagertreiber auf jedem ESXi-Host anhand einer der folgenden Vorgehensweisen:

  - Installieren Sie den Treiber auf jedem Host und fügen Sie die VIB-Datei des Treibers zum vSphere Lifecycle-Image für den Cluster hinzu.

    Weitere Informationen finden Sie in der Kurzanleitung für NVIDIA Virtual GPU-Software.

  - Fügen Sie die VIB-Datei des Treibers zum vSphere Lifecycle-Image für den Cluster hinzu und standardisieren Sie die Hosts.

  c Wenn Sie die Option „GPU mit mehreren Instanzen (MIG)“ verwenden möchten, aktivieren Sie sie auf jedem ESXi-Host im Cluster.

  Weitere Informationen finden Sie unter NVIDIA MIG-Benutzerhandbuch.

d Legen Sie auf der vCenter Server-Instanz für die VI-Arbeitslastdomäne die erweiterte `vgpu.hotmigrate.enabled`-Einstellung auf `true` fest, damit virtuelle Maschinen mit vGPU mithilfe von vSphere vMotion migriert werden können.

Weitere Informationen finden Sie unter Konfigurieren von erweiterten Einstellungen.

Erstellen einer Inhaltsbibliothek mit Deep Learning-VM-Images für VMware Private AI Foundation with NVIDIA

Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA werden in einer gemeinsam genutzten Inhaltsbibliothek verteilt, die von VMware veröffentlicht wird. Als Cloud-Administrator verwenden Sie eine Inhaltsbibliothek, um während der VM-Bereitstellung bestimmte VM-Images in Ihrer VI-Arbeitslastdomäne abzurufen.

Voraussetzungen

Stellen Sie als Cloud-Administrator sicher, dass VMware Private AI Foundation with NVIDIA bereitgestellt und konfiguriert ist. Weitere Informationen finden Sie unter Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten.

Prozedur

1 Melden Sie sich bei der vCenter Server-Instanz für die VI-Arbeitslastdomäne unter **`https://<vcenter_server_fqdn>/ui`** an.

2 Wählen Sie **Menü > Inhaltsbibliotheken** aus und klicken Sie auf **Erstellen**.

3 Erstellen Sie eine Inhaltsbibliothek für die Deep Learning-VM-Images.

- Erstellen Sie für eine verbundene Umgebung eine abonnierte Inhaltsbibliothek, die mit `https://packages.vmware.com/dl-vm/lib.json` verbunden ist. Authentifizierung ist nicht erforderlich.
- Laden Sie in einer nicht verbundenen Umgebung die Deep Learning-VM-Images von `https://packages.vmware.com/dl-vm/` herunter und importieren Sie sie in eine lokale Inhaltsbibliothek.

  Laden Sie für jedes Image die relevanten `.ovf`-, `.vmdk`-, `.mf`- und `.cert`-Dateien herunter.

Konfigurieren von vSphere IaaS Control Plane für VMware Private AI Foundation with NVIDIA

Um DevOps-Ingenieuren und Datenwissenschaftlern die Möglichkeit zu geben, Deep Learning-VMs oder TKG-Cluster mit KI-Containerarbeitslasten bereitzustellen, müssen Sie einen Supervisor auf einem GPU-fähigen Cluster in einer VI-Arbeitslastdomäne bereitstellen und vGPU-fähige VM-Klassen erstellen.

Voraussetzungen

Siehe Anforderungen für die Bereitstellung von VMware Private AI Foundation with NVIDIA.

Verfahren

1 Stellen Sie einen NSX Edge-Cluster in der VI-Arbeitslastdomäne mithilfe von SDDC Manager bereit.

   SDDC Manager stellt auch ein Tier-0-Gateway bereit, das Sie bei der Supervisor-Bereitstellung angeben. Das Tier-0-Gateway befindet sich im Aktiv/Aktiv-Hochverfügbarkeitsmodus.

2 Konfigurieren Sie eine Speicherrichtlinie für den Supervisor.

   Weitere Informationen finden Sie unter Erstellen von Speicherrichtlinien für vSphere with Tanzu.

3 Stellen Sie einen Supervisor auf einem Cluster von GPU-fähigen ESXi-Hosts in der VI-Arbeitslastdomäne bereit.

   Sie verwenden die Zuweisung statischer IP-Adressen für das Verwaltungsnetzwerk. Weisen Sie das Supervisor-VM-Verwaltungsnetzwerk auf dem vSphere Distributed Switch für den Cluster zu.

   Konfigurieren Sie das Arbeitslastnetzwerk wie folgt:

   - Verwenden Sie die vSphere Distributed Switch für den Cluster oder erstellen Sie einen speziell für KI-Arbeitslasten.
   - Konfigurieren Sie den Supervisor mit dem NSX Edge-Cluster und dem Tier-0-Gateway, das Sie mithilfe von SDDC Manager bereitgestellt haben.
   - Legen Sie die restlichen Werte entsprechend Ihrer Einrichtung fest.

   Verwenden Sie die von Ihnen erstellte Speicherrichtlinie.

   Weitere Informationen zum Bereitstellen eines Supervisors in einem einzelnen Cluster finden Sie unter Bereitstellen eines Supervisors für eine Zone mit NSX-Netzwerk.

4 Konfigurieren Sie vGPU-basierte VM-Klassen für KI-Arbeitslasten.

   In diesen VM-Klassen legen Sie die Computing-Anforderungen und ein vGPU-Profil für ein NVIDIA GRID vGPU-Gerät entsprechend den vGPU-Geräten fest, die auf den ESXi-Hosts im Supervisor-Cluster konfiguriert sind.

   - Informationen zum Einrichten von vGPU-basierten VM-Klassen für virtuelle Maschinen finden Sie unter Erstellen einer benutzerdefinierten VM-Klasse mit dem vSphere Client und Hinzufügen von PCI-Geräten zu einer VM-Klasse in vSphere with Tanzu.
   - Informationen zum Einrichten von vGPU-basierten VM-Klassen für TKG-Worker-Knoten finden Sie unter Erstellen Sie eine benutzerdefinierte VM-Klasse mit einem vGPU-Profil in vSphere 8 Update 2b und höher und Konfigurieren von vSphere-Namespaces für TKG-Cluster auf Supervisor.

Legen Sie für die VM-Klasse für die Bereitstellung von Deep Learning-VMs mit NVIDIA RAG-Arbeitslasten die folgenden zusätzlichen Einstellungen im Dialogfeld „VM-Klasse“ fest:

- Wählen Sie das vollständige vGPU-Profil für den Zeitaufteilungsmodus oder ein MIG-Profil aus. Wählen Sie beispielsweise für die NVIDIA A100-40-GB-Karte im vGPU-Zeitaufteilungsmodus die Option **nvidia_a100-40c** aus.
- Teilen Sie auf der Registerkarte **Virtuelle Hardware** mehr als 16 virtuelle CPU-Kerne und 64 GB virtuellen Arbeitsspeicher zu.
- Legen Sie auf der Registerkarte **Erweiterte Parameter** den Parameter `pciPassthru<vgpu-id>.cfg.enable_uvm` auf **1** fest.

  wobei `<vgpu-id>` die der virtuellen Maschine zugewiesene vGPU identifiziert. Wenn der virtuellen Maschine beispielsweise zwei vGPUs zugewiesen sind, legen Sie `pciPassthru0.cfg.parameter=1` und `pciPassthru1.cfg.parameter = 1` fest.

5 Wenn Sie das `kubectl`-Befehlszeilentool verwenden möchten, um eine Deep Learning-VM oder einen GPU-beschleunigten TKG-Cluster auf einem Supervisor bereitzustellen, erstellen und konfigurieren Sie einen vSphere-Namespace, fügen Sie Ressourcengrenzwerte, Speicherrichtlinien, Berechtigungen für DevOps-Ingenieure hinzu und verknüpfen Sie die vGPU-basierten VM-Klassen damit.

- Informationen zum Einrichten von vSphere-Namespaces für virtuelle Maschinen finden Sie unter Erstellen und Konfigurieren eines vSphere-Namespace auf dem Supervisor.
- Informationen zum Einrichten von vSphere-Namespaces für TKG-Cluster finden Sie unter Konfigurieren von vSphere-Namespaces für TKG-Cluster auf Supervisor.

6 Wenn Sie Bereitstellungen von Deep Learning-VMs auf einem Supervisor aktivieren möchten, indem Sie `kubectl` direkt aufrufen, fügen Sie die Inhaltsbibliothek zum vSphere-Namespace für KI-Arbeitslasten hinzu.

VMware Aria Automation erstellt jedes Mal, wenn eine Deep Learning-VM bereitgestellt wird, einen Namespace und fügt ihr automatisch die Inhaltsbibliothek hinzu.

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

b Navigieren Sie zum Namespace für KI-Arbeitslasten.

c Klicken Sie auf der Karte **VM-Dienst** auf **Inhaltsbibliotheken verwalten**.

d Wählen Sie die Inhaltsbibliothek mit den Deep Learning-VM-Images aus und klicken Sie auf **OK**.

Konfigurieren einer Inhaltsbibliothek mit Ubuntu TKr für eine getrennte VMware Private AI Foundation with NVIDIA-Umgebung

Wenn Ihre Umgebung über keine Internetverbindung verfügt, stellen Sie als Cloud-Administrator eine lokale Inhaltsbibliothek bereit, in die Sie Tanzu Kubernetes-Versionen (TKr) manuell hochladen und mit dem Supervisor verknüpfen.

Die Bereitstellung von NVIDIA-fähigen KI-Arbeitslasten auf TKG-Clustern erfordert die Verwendung der Ubuntu-Edition von Tanzu Kubernetes-Versionen.

Vorsicht Die TKr-Inhaltsbibliothek wird in allen vSphere-Namespaces im Supervisor verwendet, wenn Sie neue TKG-Cluster bereitstellen.

Voraussetzungen

Stellen Sie als Cloud-Administrator sicher, dass VMware Private AI Foundation with NVIDIA bereitgestellt und konfiguriert ist. Siehe Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten.

Prozedur

1 Laden Sie die Ubuntu-basierten TKr-Images mit den erforderlichen Kubernetes-Versionen von `https://wp-content.vmware.com/v2/latest/` herunter.

2 Melden Sie sich bei der vCenter Server-Instanz für die VI-Arbeitslastdomäne unter **`http://<vcenter_server_fqdn>/ui`** an.

3 Erstellen Sie eine lokale Inhaltsbibliothek und importieren Sie die TKr-Images dort.

   Weitere Informationen finden Sie unter Erstellen einer lokalen Inhaltsbibliothek (für die Bereitstellung von Air-Gapped-Clustern).

4 Fügen Sie die Inhaltsbibliothek zum Supervisor hinzu.

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

   b Navigieren Sie zum Supervisor für KI-Arbeitslasten.

   c Wählen Sie auf der Registerkarte **Konfigurieren** die Option **Allgemein** aus.

   d Klicken Sie neben der Eigenschaft **Tanzu Kubernetes Grid Service** auf **Bearbeiten**.

   e Erweitern Sie auf der angezeigten Seite **Allgemein** die Option **Tanzu Kubernetes Grid Service** und klicken Sie neben **Inhaltsbibliothek** auf **Bearbeiten**.

   f Wählen Sie die Inhaltsbibliothek mit den TKr-Images aus und klicken Sie auf **OK**.

Einrichten einer Private Harbor-Registrierung in VMware Private AI Foundation with NVIDIA

Sie können Harbor als Supervisor-Dienst und als lokale Registrierung für Container-Images aus dem NVIDIA NGC-Katalog verwenden.

Hinweis Für die Installation des Harbor-Diensts im Supervisor ist eine Internetverbindung erforderlich.

Wenn Sie die Integration der Harbor-Registrierung mit Supervisor verwenden möchten, können Sie die folgenden Einrichtungsmethoden anwenden:

- Verwenden Sie eine Harbor-Registrierung nur im Supervisor in der GPU-fähigen Arbeitslastdomäne. Führen Sie die folgenden Aufgaben aus:

  a Aktivieren von Harbor als ein Supervisor-Dienst.

  b Hochladen von KI-Container-Images in eine Private Harbor-Registrierung in VMware Private AI Foundation with NVIDIA

  Sie können Ihre Umgebung vom Internet trennen und den Harbor-Dienst nach der Installation des Dienstes oder nach der Installation und dem Herunterladen des ersten Satzes der erforderlichen Container-Images als lokale Container-Registrierung verwenden.

  Bei dieser Methode müssen Sie Container-Images manuell aus dem NVIDIA NGC-Katalog auf eine Maschine in der Umgebung herunterladen und dann in die Registrierung hochladen.

- Erstellen einer Harbor-Registrierung in VMware Private AI Foundation with NVIDIA als Replikat einer verbundenen Registrierung.

  Eine Harbor-Registrierung, die außerhalb der VMware Private AI Foundation with NVIDIA-Umgebung ausgeführt wird, ist immer mit dem Internet verbunden. Die Harbor-Registrierung im Supervisor für die GPU-fähige Arbeitslastdomäne empfängt Container-Images von der verbundenen Domäne mithilfe eines Proxy-Mechanismus. Auf diese Weise bleiben die Hauptkomponenten der VMware Cloud Foundation-Instanz isoliert.

  Bei diesem Ansatz sind zusätzliche Ressourcen für die verbundene Registrierung erforderlich.

Hinweis Teilen Sie ausreichend Speicherplatz für das Hosting der NVIDIA NGC-Container zu, die Sie auf einer Deep Learning-VM oder einem TKG-Cluster bereitstellen möchten. Sie sollten mindestens drei Versionen jedes Containers im Speicher bereitstellen.

Wenn die Verbindung mit dem Internet während der Installation des Harbor-Diensts oder der Einrichtung einer verbundenen Harbor-Registrierung für Ihre Organisation keine Option ist, verwenden Sie eine Containerregistrierung eines anderen Anbieters.

Hochladen von KI-Container-Images in eine Private Harbor-Registrierung in VMware Private AI Foundation with NVIDIA

In einer nicht verbundenen Umgebung, in der Sie eine Harbor-Registrierung nur auf dem KI-fähigen Supervisor verwenden, müssen Sie die KI-Container-Images, die Sie auf einer Deep Learning-VM oder einem TKG-Cluster bereitstellen möchten, manuell aus dem NVIDIA NGC-Katalog in Harbor hochladen.

Verfahren

1 Konfigurieren Sie auf den Maschinen mit Zugriff auf NVIDIA NGC und auf die getrennte VMware Cloud Foundation-Instanz den Docker-Client mit dem Zertifikat der Harbor-Registrierung.

   Siehe Konfigurieren eines Docker-Clients mit einem Registrierungszertifikat.

2 Melden Sie sich bei NVIDA NGC an.

   Verwenden Sie den reservierten Benutzernamen von $oauthtoken und fügen Sie den API-Schlüssel in das Kennwortfeld ein.

   ```
   docker login nvcr.io
   ```

3 Ziehen Sie die erforderlichen Container-Images auf die Maschine mit Zugriff auf den NVIDIA NGC-Katalog und speichern Sie sie in einem Archiv.

   Beispiel: Zum Herunterladen des CUDA-Beispiel-Container-Images führen Sie folgende Befehle aus.

   ```
   docker pull nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8
   docker save > cuda-sample.tar nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8
   ```

4 Kopieren Sie das Archiv auf die Maschine mit Zugriff auf die lokale Containerregistrierung.

5 Laden Sie auf der Maschine mit Zugriff auf die lokale Containerregistrierung das Container-Image.

   ```
   docker load < cuda-sample.tar
   ```

6 Melden Sie sich bei der Harbor-Registrierung an.

   Wenn die Harbor-Registrierung beispielsweise unter my-harbor-registry.example.com ausgeführt wird, verwenden Sie die folgenden Befehle.

   ```
   docker login my-harbor-registry.example.com
   ```

7 Taggen Sie das Image, das Sie an das Projekt weitergeben möchten, mit demselben Namen wie den zu verwendenden Namespace.

   Beispiel: Führen Sie folgenden Befehl aus, um das CUDA-Beispiel-Container-Image für das Projekt „my-private-ai-namespace“ in der Registrierung „my-harbor-registry.example.com“ als aktuelles Image zu kennzeichnen.

```
docker tag nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8 my-harbor-registry.example.com/my-private-ai-namespace/cuda-sample:latest
```

8 Übertragen Sie die Container-Images an die Harbor-Registrierung.

```
docker push my-harbor-registry.example.com/my-private-ai-namespace/cuda-sample:latest
```

Erstellen einer Harbor-Registrierung in VMware Private AI Foundation with NVIDIA als Replikat einer verbundenen Registrierung

Um die neuesten Images im NVIDIA NGC-Katalog problemlos zu aktualisieren, können Sie eine Harbor-Registrierung in einem Supervisor verwenden, der sich in einer anderen VI-Arbeitslastdomäne oder VMware Cloud Foundation-Instanz befindet und mit dem Internet verbunden ist. Anschließend replizieren Sie diese verbundene Registrierung auf dem Supervisor, auf dem Sie KI-Arbeitslasten ausführen möchten.

Sie rufen die neuesten Container-Images von NVIDIA NGC in die verbundene Harbor-Registrierung ab und übertragen sie mithilfe einer Proxy-Cache-Verbindung in die getrennte Harbor-Registrierung. Auf diese Weise müssen Sie Container-Images nicht häufig herunterladen und dann manuell hochladen.

Hinweis Sie können auch eine verbundene Containerregistrierung eines anderen Anbieters verwenden.

Sie richten das Netzwerk zwischen den beiden Registrierungen wie folgt ein:

- Die verbundene Registrierung kann zur Replikatregistrierung geroutet werden.
- Die verbundene Registrierung wird in einer DMZ platziert, in der nur `docker push`- und `docker pull`-Kommunikation zwischen den beiden Registrierungen zulässig ist.

Voraussetzungen

Aktivieren von Harbor als ein Supervisor-Dienst im Supervisor in der GPU-fähigen Arbeitslastdomäne.

Verfahren

1 Melden Sie sich bei der verbundenen Benutzeroberfläche der Harbor-Registrierung als Harbor-Systemadministrator an.

2 Wechseln Sie zur Seite **Verwaltung > Registrierungen**, erstellen Sie einen Endpoint für den NVIDIA NGC-Katalog `nvcr.io/nvaie` und wählen Sie den Anbieter **Docker-Registrierung** sowie Ihren NVIDIA NGC-API-Schlüssel aus.

3 Wechseln Sie zur Seite **Verwaltung > Projekte** und erstellen Sie ein Proxy-Cache-Projekt, das mit dem Endpoint für `nvcr.io/nvaie` verbunden ist.

4 Erstellen Sie auf der Seite **Registrierungen** einen Replizierungs-Endpoint für die getrennte Registrierung und wählen Sie den Anbieter **Harbor**.

5 Wechseln Sie zur Seite **Verwaltung > Replizierungen** und erstellen Sie eine Replizierungsregel.

   - Verwenden Sie den Push-basierten Replizierungsmodus.
   - Geben Sie in der Eigenschaft **Zielregistrierung** die URL der getrennten Registrierung auf dem KI-fähigen Supervisor ein.
   - Legen Sie Filter, Ziel-Namespace und Auslösermodus entsprechend den Anforderungen Ihrer Organisation fest.

Nächste Schritte

1 Ziehen Sie die Container-Images, die von Ihrer Organisation benötigt werden, von NVIDIA NGC in die verbundene Registrierung, indem Sie `docker pull` auf dem Docker-Clientcomputer ausführen.

2 Wenn die Replizierungsregel den manuellen Auslösermodus aufweist, führen Sie Replizierungen nach Bedarf manuell aus.

Hochladen der NVIDIA GPU-Operatorkomponenten in eine getrennte Umgebung

Laden Sie in einer getrennten Umgebung die Komponenten des NVIDIA GPU-Operators in interne Speicherorte hoch.

Verfahren

1 Stellen Sie ein lokales Ubuntu-Paket-Repository bereit und laden Sie die Container-Images im NVIDIA GPU-Operator-Paket in die Harbor-Registrierung für den Supervisor hoch.

2 Stellen Sie ein lokales Helm-Diagramm-Repository mit NVIDIA GPU-Operator-Diagrammdefinitionen bereit.

3 Aktualisieren Sie die Helm-Diagramm-Definitionen des NVIDIA GPU-Operators, um das lokale Ubuntu-Paket-Repository und die private Harbor-Registrierung zu verwenden.

Ergebnisse

Weitere Informationen finden Sie unter Installieren von VMware vSphere with VMware Tanzu (Air-Gap).

Einrichten von VMware Aria Automation für VMware Private AI Foundation with NVIDIA

VMware Aria Automation bietet Unterstützung für Self-Service-Katalogelemente, die DevOps-Ingenieure und Datenwissenschaftler verwenden können, um KI-Arbeitslasten auf benutzerfreundliche und anpassbare Weise in VMware Private AI Foundation with NVIDIA bereitzustellen.

Voraussetzungen

Stellen Sie als Cloud-Administrator sicher, dass die VMware Private AI Foundation with NVIDIA-Umgebung konfiguriert ist. Weitere Informationen finden Sie unter Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten.

Verfahren

1. Verbinden von VMware Aria Automation mit einer Arbeitslastdomäne für VMware Private AI Foundation with NVIDIA

   Bevor Sie die Katalogelemente für die Bereitstellung von KI-Anwendungen mithilfe von VMware Aria Automation hinzufügen können, verbinden Sie VMware Aria Automation mit VMware Cloud Foundation.

2. Erstellen von KI-Self-Service-Katalogelementen in VMware Aria Automation

   Als Cloud-Administrator verwenden Sie den Assistenten für die Katalogeinrichtung für Private AI in VMware Aria Automation, um schnell Katalogelemente für die Bereitstellung von Deep Learning-VMs oder GPU-beschleunigten TKG-Clustern zu einer VI-Arbeitslastdomäne in der verbundenen VMware Cloud Foundation-Instanz hinzuzufügen.

3. Erstellen eines Vektordatenbank-Katalogelements in VMware Aria Automation

   Fügen Sie als Cloud-Administrator ein Katalogelement für die Bereitstellung von Datenbanken in VMware Data Services Manager zum Automation Service Broker in VMware Aria Automation hinzu.

Verbinden von VMware Aria Automation mit einer Arbeitslastdomäne für VMware Private AI Foundation with NVIDIA

Bevor Sie die Katalogelemente für die Bereitstellung von KI-Anwendungen mithilfe von VMware Aria Automation hinzufügen können, verbinden Sie VMware Aria Automation mit VMware Cloud Foundation.

Verfahren

- Führen Sie auf der VMware Aria Automation-Benutzeroberfläche den Schnellstartassistenten für VMware Cloud Foundation oder VMware vCenter Server aus.

  Weitere Informationen finden Sie unter Vorgehensweise für die ersten Schritte mit VMware Aria Automation mithilfe des VMware Cloud Foundation-Schnellstarts oder Vorgehensweise für die ersten Schritte mit VMware Aria Automation mithilfe des VMware vCenter Server-Schnellstarts in der Dokumentation *Erste Schritte mit VMware Aria Automation*.

Erstellen von KI-Self-Service-Katalogelementen in VMware Aria Automation

Als Cloud-Administrator verwenden Sie den Assistenten für die Katalogeinrichtung für Private AI in VMware Aria Automation, um schnell Katalogelemente für die Bereitstellung von Deep Learning-VMs oder GPU-beschleunigten TKG-Clustern zu einer VI-Arbeitslastdomäne in der verbundenen VMware Cloud Foundation-Instanz hinzuzufügen.

Datenwissenschaftler können Deep Learning-Katalogelemente für die Bereitstellung von Deep Learning-VMs verwenden. DevOps-Ingenieure können die Katalogelemente für die Bereitstellung von KI-fähigen TKG-Clustern verwenden.

Bei jeder Ausführung fügt der Assistent für die Katalogeinrichtung für Private AI Elemente für Deep Learning-VMs und TKG-Cluster zum Service Broker-Katalog hinzu. Sie können den Assistenten jedes Mal ausführen, wenn Sie Folgendes benötigen:

- Aktivieren der Bereitstellung von KI-Arbeitslasten auf einem anderen Supervisor.
- Berücksichtigen einer Änderung Ihrer NVIDIA AI Enterprise-Lizenz, einschließlich der `.tok`-Datei für die Clientkonfiguration und des Lizenzservers oder der Download-URL für die vGPU-Gasttreiber für eine getrennte Umgebung.
- Integrieren einer Deep Learning-VM-Image-Änderung.
- Verwenden anderer vGPU- oder Nicht-GPU-VM-Klassen, Speicherrichtlinien oder Containerregistrierungen.
- Erstellen von Katalogelementen in einem neuen Projekt.

Hinweis Bei jeder Bereitstellung einer Deep Learning-VM oder eines Tanzu Kubernetes Grid-Clusters erstellt VMware Aria Automation einen vSphere-Namespace.

Verfahren

- Hinzufügen von Private AI-Elementen zum Automation Service Broker-Katalog.

Nächste Schritte

Mithilfe des Automation Service Broker können Ihre Datenwissenschaftler mit der Bereitstellung von Deep Learning-VMs und Ihre DevOps-Ingenieure mit der Bereitstellung GPU-fähiger Tanzu Kubernetes Grid-Cluster fortfahren. Weitere Informationen finden Sie unter Bereitstellen einer Deep Learning-VM ohne RAG in VMware Aria Automation.

Erstellen eines Vektordatenbank-Katalogelements in VMware Aria Automation

Fügen Sie als Cloud-Administrator ein Katalogelement für die Bereitstellung von Datenbanken in VMware Data Services Manager zum Automation Service Broker in VMware Aria Automation hinzu.

Voraussetzungen

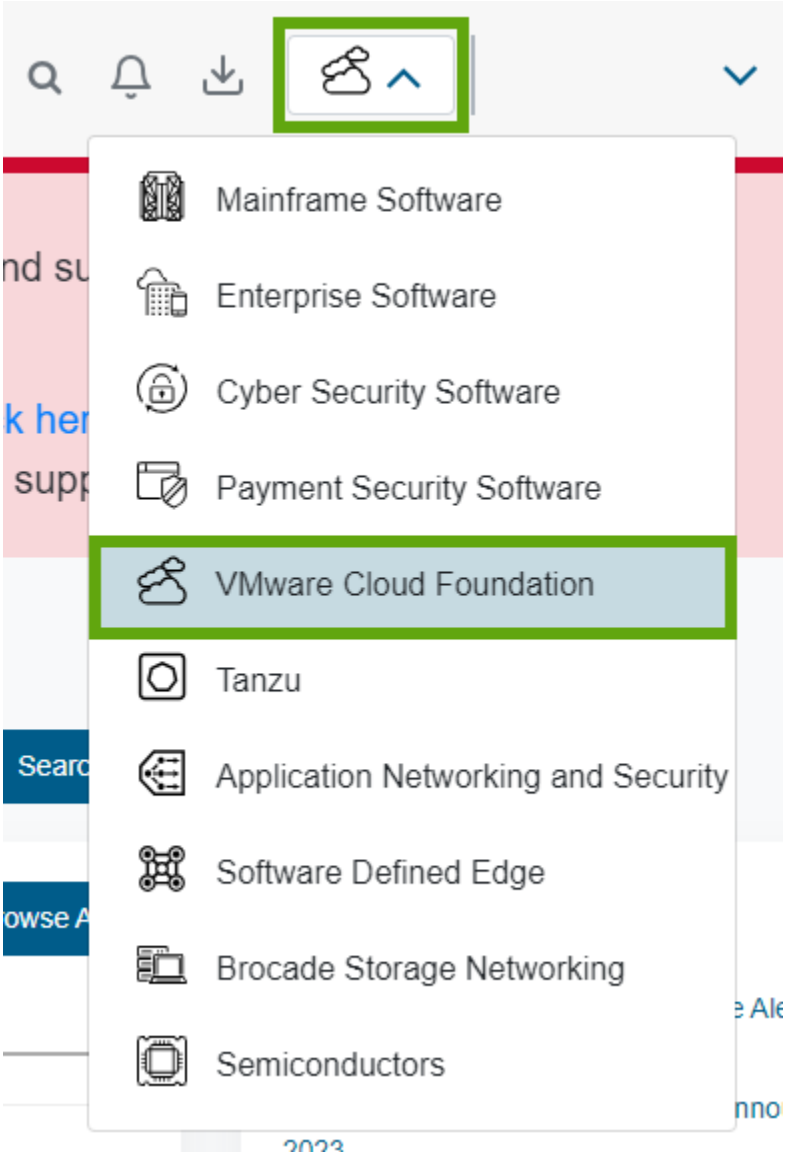
- Stellen Sie sicher, dass VMware Data Services Manager 2.1 bereitgestellt wurde.
- Stellen Sie eine Maschine bereit, auf der Python 3.10 installiert ist und auf die VMware Data Services Manager- und VMware Aria Automation-Instanzen zugreifen können.

Verfahren

1 Laden Sie das Paket `AriaAutomation_DataServicesManager` für VMware Data Services Manager 2.1 aus dem Broadcom Technical Portal herunter.

   a Melden Sie sich beim Broadcom-Supportportal an.

   b Wählen Sie im Dropdown-Menü für die Softwarekategorie in der oberen rechten Ecke des Portals die Option **VMware Cloud Foundation** aus.



   c Klicken Sie im linken Navigationsbereich auf **Meine Downloads**.

   d Klicken Sie auf der Seite **Meine Downloads – VMware Cloud Foundation** auf **VMware Data Services Manager**.

   e Klicken Sie auf die Versionsnummer und laden Sie das Paket `AriaAutomation_DataServicesManager` herunter.

2 Laden Sie auf der Maschine unter Python das Paket `AriaAutomation_DataServicesManager` hoch und extrahieren Sie dessen Inhalt.

3 Aktualisieren Sie die `config.json`-Datei in dem Ordner, in dem Sie das Paket extrahiert haben, mit den URLs und Benutzeranmeldedaten für VMware Data Services Manager und VMware Aria Automation.

   Optional können Sie auch den Namen des Katalogelements, das Automation Assembler-Projekt und andere Parameter festlegen.

4 Um die Katalogelemente in VMware Aria Automation zu erstellen, führen Sie das Python-Skript `aria.py` folgendermaßen aus.

```
python3 aria.py enable-blueprint-version-2
```

Ergebnisse

Das Python-Skript erstellt Elemente in VMware Aria Automation, die für die Verwendung von VMware Data Services Manager für die Datenbankbereitstellung erforderlich sind. Weitere Informationen finden Sie in der `readme.md`-Datei im `AriaAutomation_DataServicesManager`-Paket

Nächste Schritte

Ihre Datenwissenschaftler oder DevOps-Ingenieure können eine Vektordatenbank aus dem Automation Service Broker-Katalog mit der Erweiterung „pgvector“ bereitstellen und die zugehörigen RAG-Arbeitslasten integrieren. Weitere Informationen finden Sie unter Kapitel 5 Bereitstellen von RAG-Arbeitslasten in VMware Private AI Foundation with NVIDIA.

Bereitstellen einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA

3

VMware Private AI Foundation with NVIDIA unterstützt die Bereitstellung vorkonfigurierter Deep Learning-VMs, die Datenwissenschaftler für die KI-Entwicklung verwenden können.

Als Datenwissenschaftler haben Sie folgende Möglichkeiten, um mit der Verwendung einer Deep Learning-VM zu beginnen:

- Stellen Sie eine Deep Learning-VM mithilfe eines Self-Service-Katalogelements in VMware Aria Automation bereit.
- Bitten Sie Ihren DevOps-Ingenieur, mithilfe des Befehls `kubectl` eine Deep Learning-VM auf einem Tanzu Kubernetes Grid-Cluster bereitzustellen.
- Bitten Sie Ihren Cloud-Administrator, eine Deep Learning-VM auf einem vSphere-Cluster bereitzustellen, damit Sie sich schnell mit den Deep Learning-VM-Vorlagen vertraut machen können.
- Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA

  Die im Rahmen von VMware Private AI Foundation with NVIDIA bereitgestellten Deep Learning-VM-Images sind mit gängigen ML-Bibliotheken, Frameworks und Toolkits vorkonfiguriert und werden von NVIDIA und VMware für die GPU-Beschleunigung in einer VMware Cloud Foundation-Umgebung optimiert und validiert.
- Bereitstellen einer Deep Learning-VM mithilfe eines Self-Service-Katalogs in VMware Aria Automation

  In VMware Private AI Foundation with NVIDIA können Sie als Datenwissenschaftler oder DevOps-Ingenieur eine Deep Learning-VM aus VMware Aria Automation bereitstellen, indem Sie Self-Service-Katalogelemente einer KI-Workstation in Automation Service Broker verwenden.
- Direktes Bereitstellen einer Deep Learning-VM auf einem vSphere-Cluster in VMware Private AI Foundation with NVIDIA

  Damit Datenwissenschaftler die Deep Learning-VM-Vorlagen in VMware Private AI Foundation with NVIDIA schnell testen können, können Sie als Cloud-Administrator mithilfe des vSphere Client eine Deep Learning-VM direkt auf einem vSphere-Cluster bereitstellen.

- Bereitstellen einer Deep Learning-VM mithilfe des Befehls „`kubectl`“ in VMware Private AI Foundation with NVIDIA

  Der VM-Dienst im Supervisor in vSphere IaaS Control Plane ermöglicht DevOps-Ingenieuren die Bereitstellung und Ausführung von Deep Learning-VMs mithilfe der Kubernetes-API.
- Anpassen der Bereitstellung von Deep Learning-VMs in VMware Private AI Foundation with NVIDIA

  Wenn Sie eine Deep Learning-VM in vSphere IaaS control plane mithilfe von `kubectl` oder direkt auf einem vSphere-Cluster bereitstellen, müssen Sie benutzerdefinierte VM-Eigenschaften eingeben.
- Fehlerbehebung bei der Bereitstellung von Deep Learning-VMs in VMware Private AI Foundation with NVIDIA

  Die Fehlerbehebungsinformationen zur Bereitstellung einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA enthalten Lösungen für potenzielle Probleme, die auftreten können.

Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA

Die im Rahmen von VMware Private AI Foundation with NVIDIA bereitgestellten Deep Learning-VM-Images sind mit gängigen ML-Bibliotheken, Frameworks und Toolkits vorkonfiguriert und werden von NVIDIA und VMware für die GPU-Beschleunigung in einer VMware Cloud Foundation-Umgebung optimiert und validiert.

Als Datenwissenschaftler können Sie die über diese Images bereitgestellten Deep Learning-VMs für KI-Prototyperstellung, Feinabstimmung, Validierung und Rückschlüsse verwenden.

Der Software-Stack für die Ausführung von KI-Anwendungen auf NVIDIA GPUs wird im Voraus validiert. Folglich beginnen Sie direkt mit der KI-Entwicklung, ohne Zeit für die Installation und Validierung der Kompatibilität von Betriebssystemen, Softwarebibliotheken, ML-Frameworks, Toolkits und GPU-Treibern aufwenden zu müssen.

Was enthält ein Deep Learning-VM-Image?

Das aktuelle Image der Deep Learning-VM enthält die folgende Software. Informationen zu den Komponentenversionen in jeder Deep Learning-VM-Image-Version finden Sie unter Versionshinweise zu VMware Deep Learning VM.

Softwarekomponentenkategorie	Softwarekomponente	
Eingebettet	■ Kanonisches Ubuntu ■ NVIDIA-Container-Toolkit ■ Docker-Community-Engine ■ Miniconda und ein PyTorch Conda-Manifest.	
Kann automatisch vorinstalliert werden, wenn Sie die Deep Learning-VM erstmalig starten	■ vGPU-Gasttreiber entsprechend der Version des vGPU-Hosttreibers	
	DL-Arbeitslasten (Deep Learning)	CUDA-Beispiel Sie können eine Deep Learning-VM mit ausgeführten CUDA-Beispielen verwenden, um die Vektorhinzufügung, die Gravitations-N-Körper-Simulation oder andere Beispiele auf einer VM zu untersuchen. Weitere Informationen finden Sie auf der Seite CUDA-Beispiele im NVIDIA NGC-Katalog.
		PyTorch. Sie können eine Deep Learning-VM mit einer PyTorch-Bibliothek verwenden, um Konversations-KI, NLP und andere Arten von KI-Modellen auf einer VM zu erkunden. Weitere Informationen finden Sie auf der Seite PyTorch im NVIDIA NGC-Katalog. Sie können eine einsatzbereite JupyterLab-Instanz mit installiertem und konfiguriertem PyTorch unter `http://dl_vm_ip:8888` verwenden.
		TensorFlow. Sie können eine Deep Learning-VM mit einer TensorFlow-Bibliothek verwenden, um Konversations-KI, NLP und andere Arten von KI-Modellen auf einer VM zu erkunden. Weitere Informationen finden Sie auf der Seite TensorFlow im NVIDIA NGC-Katalog. Sie können eine betriebsbereite JupyterLab-Instanz mit installiertem und konfiguriertem TensorFlow unter `http://dl_vm_ip:8888` verwenden.
		DCGM Exporter Sie können einer Deep Learning-VM mit einem DCGM Exporter (Data Center GPU Manager) verwenden, um den Zustand von GPUs zu überwachen und Metriken aus GPUs abzurufen, die von einer DL-Arbeitslast verwendet werden, indem Sie NVIDIA DCGM, Prometheus und Grafana verwenden. Weitere Informationen finden Sie auf der Seite DCGM Exporter im NVIDIA NGC-Katalog. In einer Deep Learning-VM führen Sie den DCGM Exporter-Container zusammen mit einer DL-Arbeitslast aus, die KI-Vorgänge durchführt. Nachdem die Deep Learning-VM gestartet wurde, ist DCGM Exporter bereit, vGPU-Metriken zu erfassen und die Daten zur weiteren Überwachung und Visualisierung in eine andere Anwendung zu exportieren. Informationen zur Verwendung von DGCM Exporter zum Visualisieren von Metriken mit Prometheus und Grafana finden Sie unter DCGM Exporter.

Softwarekomponentenkategorie	Softwarekomponente	
		Triton Inference Server Sie können eine Deep Learning-VM mit einem Triton Inference Server verwenden, um ein Modell-Repository zu laden und Rückschlussanforderungen zu erhalten. Weitere Informationen finden Sie auf der Seite Triton Inference Server im NVIDIA NGC-Katalog. Informationen zur Verwendung des Triton Inference Servers zum Anfordern von Inferenzen für KI-Modelle finden Sie unter Triton Inference Server.
		NVIDIA RAG Sie können eine Deep Learning-VM verwenden, um RAG-Lösungen (Retrieval Augmented Generation) mit einem Llama2-Modell zu erstellen. Weitere Informationen finden Sie in der Dokumentation zu NVIDIA RAG-Anwendungen mit Docker Compose (erfordert bestimmte Kontoberechtigungen). Ein Beispiel für eine Chatbot-Webanwendung, auf die Sie unter `http://dl_vm_ip:3001/orgs/nvidia/models/text-qa-chatbot` zugreifen können. Sie können Ihre eigene Knowledgebase hochladen.

Bereitstellen einer Deep Learning-VM

Als Datenwissenschaftler können Sie selbst eine Deep Learning-VM bereitstellen, indem Sie Katalogelemente in VMware Aria Automation verwenden. Ansonsten stellt ein Cloud-Administrator oder DevOps-Ingenieur eine solche VM für Sie bereit.

Bereitstellen einer Deep Learning-VM mithilfe eines Self-Service-Katalogs in VMware Aria Automation

In VMware Private AI Foundation with NVIDIA können Sie als Datenwissenschaftler oder DevOps-Ingenieur eine Deep Learning-VM aus VMware Aria Automation bereitstellen, indem Sie Self-Service-Katalogelemente einer KI-Workstation in Automation Service Broker verwenden.

Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA finden Sie unter Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA.

Voraussetzungen

- Stellen Sie sicher, dass Ihr Cloud-Administrator den VMware Aria Automation-Katalog für die Bereitstellung der Private AI-Anwendung eingerichtet hat. Weitere Informationen finden Sie unter Hinzufügen von Private AI-Elementen zum Automation Service Broker-Katalog.
- Stellen Sie sicher, dass Ihr Cloud-Administrator die Benutzerrolle zugewiesen hat, die für die Bereitstellung von Deep Learning-VMs erforderlich ist.

Verfahren

- Bereitstellen einer Deep Learning-VM ohne RAG in VMware Aria Automation oder Bereitstellen einer Deep Learning-VM mit einer RAG-Arbeitslast.

  Für die Bereitstellung einer Deep Learning-VM mit NVIDIA RAG ist eine Vektordatenbank erforderlich, wie z. B. eine PostgreSQL-Datenbank mit pgvector in VMware Data Services Manager.

Ergebnisse

Der vGPU-Gasttreiber und die angegebene Deep Learning-Arbeitslast werden beim ersten Start der Deep Learning-VM installiert.

Nächste Schritte

Wenn Sie Informationen zum Zugriff auf die virtuelle Maschine und die JupyterLab-Instanz benötigen, die im Lieferumfang bestimmter Deep Learning-VM-Images enthalten ist, navigieren Sie in Automation Service Broker zu **Nutzung > Bereitstellungen > Bereitstellungen**.

Direktes Bereitstellen einer Deep Learning-VM auf einem vSphere-Cluster in VMware Private AI Foundation with NVIDIA

Damit Datenwissenschaftler die Deep Learning-VM-Vorlagen in VMware Private AI Foundation with NVIDIA schnell testen können, können Sie als Cloud-Administrator mithilfe des vSphere Client eine Deep Learning-VM direkt auf einem vSphere-Cluster bereitstellen.

Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA finden Sie unter Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA.

Für die Bereitstellung einer Deep Learning-VM mit NVIDIA RAG ist eine Vektordatenbank erforderlich, wie z. B. eine PostgreSQL-Datenbank mit pgvector in VMware Data Services Manager. Informationen zum Bereitstellen einer solchen Datenbank und deren Integration in eine Deep Learning-VM finden Sie unter Bereitstellen einer Deep Learning-VM mit einer RAG-Arbeitslast.

Voraussetzungen

Stellen Sie sicher, dass VMware Private AI Foundation with NVIDIA bereitgestellt und konfiguriert ist. Weitere Informationen finden Sie unter Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten.

Verfahren

1. Melden Sie sich bei der vCenter Server-Instanz für die VI-Arbeitslastdomäne an.
2. Wählen Sie im vSphere Client-Startmenü die Option **Inhaltsbibliotheken** aus.
3. Navigieren Sie zum Deep Learning-VM-Image in der Inhaltsbibliothek.

4 Klicken Sie mit der rechten Maustaste auf eine OVF-Vorlage und wählen Sie **Neue VM über diese Vorlage** aus.

5 Geben Sie auf der Seite **Namen und Ordner auswählen** des angezeigten Assistenten einen Namen ein und wählen Sie einen VM-Ordner aus, wählen Sie **Hardware dieser virtuellen Maschine anpassen** und klicken Sie auf **Weiter**.

6 Wählen Sie einen GPU-fähigen Cluster in der VI-Arbeitslastdomäne aus, geben Sie an, ob die virtuelle Maschine nach Abschluss der Bereitstellung eingeschaltet werden muss, und klicken Sie auf **Weiter**.

7 Folgen Sie den Anweisungen des Assistenten, um einen Datenspeicher und ein Netzwerk auf dem Distributed Switch für den Cluster auszuwählen.

8 Geben Sie auf der Seite **Vorlage anpassen** die benutzerdefinierten VM-Eigenschaften ein, die zum Einrichten der KI-Funktionen erforderlich sind, und klicken Sie auf **Weiter**.

   Weitere Informationen finden Sie unter OVF-Eigenschaften von Deep Learning-VMs.

9 Weisen Sie auf der Seite **Hardware anpassen** der virtuellen Maschine ein NVIDIA vGPU-Gerät als **Neues PCI-Gerät** zu und klicken Sie auf **Weiter**.

   Wählen Sie für eine Deep Learning-VM, auf der ein NVIDIA RAG ausgeführt wird, das vGPU-Profil in voller Größe für den Zeitaufteilungsmodus oder ein MIG-Profil aus. Wählen Sie beispielsweise für NVIDIA A100 40 GB im vGPU-Zeitaufteilungsmodus die Option **nvidia_a100-40c** aus.

10 Legen Sie für eine Deep Learning-VM, auf der ein NVIDIA RAG ausgeführt wird, auf der Registerkarte **Erweitere Parameter** der Einstellungen der virtuellen Maschine den Parameter `pciPassthru<vgpu-id>.cfg.enable_uvm` auf **1** fest.

    wobei `<vgpu-id>` die der virtuellen Maschine zugewiesene vGPU identifiziert. Wenn der virtuellen Maschine beispielsweise zwei vGPUs zugewiesen sind, legen Sie `pciPassthru0.cfg.parameter=1` und `pciPassthru1.cfg.parameter = 1` fest.

11 Überprüfen Sie die Bereitstellungsspezifikation und klicken Sie auf **Beenden**.

Ergebnisse

Der vGPU-Gasttreiber und die angegebene Deep Learning-Arbeitslast werden beim ersten Start der Deep Learning-VM installiert.

Sie können die Protokolle untersuchen oder die JupyterLab-Instanz öffnen, die in einigen der Images enthalten ist. Sie können Zugriffsdetails für Datenwissenschaftler in Ihrer Organisation freigeben. Weitere Informationen finden Sie unter Deep Learning-Arbeitslasten in VMware Private AI Foundation with NVIDIA.

Nächste Schritte

- Stellen Sie über SSH eine Verbindung zur Deep Learning-VM her und stellen Sie sicher, dass alle Komponenten installiert sind und wie erwartet ausgeführt werden.
- Senden Sie die Zugriffsdetails an Ihre Datenwissenschaftler.

Bereitstellen einer Deep Learning-VM mithilfe des Befehls „`kubectl`“ in VMware Private AI Foundation with NVIDIA

Der VM-Dienst im Supervisor in vSphere IaaS Control Plane ermöglicht DevOps-Ingenieuren die Bereitstellung und Ausführung von Deep Learning-VMs mithilfe der Kubernetes-API.

Als DevOps-Ingenieur verwenden Sie `kubectl`, um eine Deep Learning-VM in dem vom Cloud-Administrator konfigurierten Namespace bereitzustellen.

Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA finden Sie unter Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA.

Für die Bereitstellung einer Deep Learning-VM mit NVIDIA RAG ist eine Vektordatenbank erforderlich, wie z. B. eine PostgreSQL-Datenbank mit pgvector in VMware Data Services Manager. Informationen zum Bereitstellen einer solchen Datenbank und deren Integration in eine Deep Learning-VM finden Sie unter Bereitstellen einer Deep Learning-VM mit einer RAG-Arbeitslast.

Voraussetzungen

Stellen Sie mithilfe des Cloud-Administrators sicher, dass VMware Private AI Foundation with NVIDIA bereitgestellt und konfiguriert ist. Weitere Informationen finden Sie unter Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten.

Verfahren

1 Melden Sie sich bei der Supervisor-Steuerungsebene an.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-
username USERNAME
```

2 Überprüfen Sie, ob alle erforderlichen VM-Ressourcen, wie z. B. VM-Klassen und VM-Images, im Namespace vorhanden sind.

Weitere Informationen finden Sie unter Anzeigen der in einem Namespace verfügbaren VM-Ressourcen in vSphere with Tanzu.

3 Bereiten Sie die YAML-Datei für die Deep Learning-VM vor.

Verwenden Sie die vm-operator-api und legen Sie die OVF-Eigenschaften als ConfigMap-Objekt fest. Informationen zu verfügbaren OVF-Eigenschaften finden Sie unter OVF-Eigenschaften von Deep Learning-VMs.

Sie können beispielsweise eine YAML-Spezifikation vom Typ `example-dl-vm.yaml` für eine Deep Learning-Beispiel-VM erstellen, auf der PyTorch in einer verbundenen Umgebung ausgeführt wird.

```
apiVersion: vmoperator.vmware.com/v1alpha1
kind: VirtualMachine
metadata:
```

```
  name: example-dl-vm
  namespace: example-dl-vm-namespace
  labels:
    app: example-dl-app
spec:
  className: gpu-a100
  imageName: vmi-xxxxxxxxxxxxx
  powerState: poweredOn
  storageClass: tanzu-storage-policy
  vmMetadata:
    configMapName: example-dl-vm-config
    transport: OvfEnv
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-dl-vm-config
  namespace: example-dl-vm-namespace
data:
  user-data:
```

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBwLnNoCiAgcGVybWlzc2lvbnM
6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbH
ZtL3V0aWxzLnNoCiAgICB0cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jrb
G9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCgogICAgREVGQVVMVF9SRUdfVVJJ
PSJudmNyLmlvIgogICAgUkVHSVNUUllfVVJJX1BBVEg9JChncmVwIHJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWV
udi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCgogICAgaWYgW1sgLXogIiRSRU
dJU1RSWV9VUklfUEFUSCIgXV07IHRoZW4KICAgICAgIyBJZiBSRUdJU1RSWV9VUklfUEFUSCBpcyBudWxsIG9yIGVtc
HR5LCB1c2UgdGhlIGRlZmF1bHQgdmFsdWUKICAgICAgUkVHSVNUUllfVVJJX1BBVEg9JERFRkFVTFRfUkVHX1VSSQog
ICAgICBlY2hvICJSRUdJU1RSWV9VUklfUEFUSCB3YXMgZW1wdHkuIFVzaW5nIGRlZmF1bHQ6ICRSRUdJU1RSWV9VUkl
fUEFUSCIKICAgIGZpCiAgICAKICAgICMgSWYgUkVHSVNUUllfVVJJX1BBVEggY29udGFpbnMgJy8nLCBleHRyYWN0IH
RoZSBVUkkgcGFydAogICAgaWYgW1sgJFJFR0lTVFJZX1VSSV9QQVRIID09ICoiLyIqIF1dOyB0aGVuCiAgICAgIFJFR
0lTVFJZX1VSST0kKGVjaG8gIiRSRUdJU1RSWV9VUklfUEFUSCIgfCBjdXQgLWQnLycgLWYxKQogICAgZWxzZQogICAg
ICBSRUdJU1RSWV9VUkk9JFJFR0lTVFJZX1VSSV9QQVRICiAgICBmaQogIAogICAgUkVHSVNUUllfVVNFUk5BTUU9JCh
ncmVwIHJlZ2lzdHJ5LXVzZXIgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW1
4iXSpcKS4qL1wxL3AnKQogICAgUkVHSVNUUllfUEFTU1dPUkQ9JChncmVwIHJlZ2lzdHJ5LXBhc3N3ZCAvb3B0L2Rsd
m0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCiAgICBpZiBbWyAt
biAiJFJFR0lTVFJZX1VTRVJOQU1FIiAmJiAtbiAiJFJFR0lTVFJZX1BBU1NXT1JEIiBdXTsgdGhlbgogICAgICBkb2N
rZXIgbG9naW4gLXUgJFJFR0lTVFJZX1VTRVJOQU1FIC1wICRSRUdJU1RSWV9QQVNTV09SRCAkUkVHSVNUUllfVVJJCi
AgICBlbHNlCiAgICAgIGVjaG8gIldhcm5pbmc6IHRoZSByZWdpc3RyeSdzIHVzZXJuYW1lIGFuZCBwYXNzd29yZCBhc
mUgaW52YWxpZCwgU2tpcHBpbmcgRG9ja2VyIGxvZ2luLiIKICAgIGZpCgogICAgZG9ja2VyIHJ1biAtZCAtLWdwdXMg
YWxsIC1wIDg4ODg6ODg4OCAkUkVHSVNUUllfVVJJX1BBVEgvbnZpZGlhL3B5dG9yY2g6MjMuMTAtcHkzIC91c3IvbG9
jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9vdCAtLWlwPSogLS1wb3J0PTg4ODggLS1uby1icm93c2VyIC0tTm
90ZWJvb2tBcHAudG9rZW49JycgLS1Ob3RlYm9va0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGlyPS93b
3Jrc3BhY2UKCi0gcGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6
IHwKICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAgICBlY2hvICJFcnJvcjogJDEiID4mMgogICA
gICB2bXRvb2xzZCAtLWNtZCAiaW5mby1zZXQgZ3Vlc3RpbmZvLnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIG
ZhbHNlLCBETFdvcmtsb2FkRmFpbHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgfQoKICAgIGNoZWNrX3Byb3RvY29sK
CkgewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQKICAgICAgbG9jYWwgc3VwcG9ydGVkX3Byb3Rv
Y29scz0oIiRAIikKICAgICAgaWYgW1sgLW4gIiR7cHJveHlfdXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm9
0b2NvbD0kKGVjaG8gIiR7cHJveHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBwcmludCAkMTsgZW
xzZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2NvbCIgXTsgdGhlbgogICAgICAgICAgZWNobyAiT
m8gc3BlY2lmaWMgcHJvdG9jb2wgcHJvdmlkZWQuIFNraXBwaW5nIHByb3RvY29sIGNoZWNrLiIKICAgICAgICAgIHJl
dHVybiAwCiAgICAgICAgZmkKICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQogICAgICAgIGZvciB
2YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwogICAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2x9Ii
A9PSAiJHt2YXJ9IiBdXTsgdGhlbgogICAgICAgICAgICBwcm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgI
```

GJyZWFrCiAgICAgICAgICBmaQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2NvbF9pbmNsdWRlZH0i
ID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0ICJVbnN1cHBvcnRlZCBwcm90b2NvbDogJHtwcm9
0b2NvbH0uIFN1cHBvcnRlZCBwcm90b2NvbHMgYXJlOiAke3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgIG
ZpCiAgICAgICAgZpCiAgICB9CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xzCiAgICBzZXRfcHJve
HkoKSB7CiAgICAgICAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIpCgogICAgICBDT05GSUdfSlNPTl9CQVNF
NjQ9JChncmVwICdjb25maWctanNvbicgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU
9IlwoW14iXSpcKS4qL1wxL3AnKQogICAgICAgICBCDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdfSlNPTl9CQVNFNjR9IH
wgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgICAgSFRUUF9QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqc
SAtciAnLmh0dHBfcHJveHkgLy8gZW1wdHknKQogICAgICAgICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19K
U09OfSIgfCBqcSAtciAnLmh0dHBzX3Byb3h5IC8vIGVtcHR5KSIgaWYgW1sgJD8gLW5lIDAgfHwgKC16ICI
ke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJPWFlfVVJMfSIpIF1dOyB0aGVuCiAgICAgICAgZWNoby
AiSW5mbzogVGhlIGNvbmZpZy1qc29uIHdhcyBwYXJzZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kL
iIKICAgICAgICAgICAgICAgICBmaQoKICAgICAgICAgICAgICAgICBmaQoKICAgICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUF9QUk9YWV9VUkx9
IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAgICAgICAgICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJ
MfSIgIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iCgogICAgICAgICAgICAgICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldG
MvZW52aXJvbm1lbnQ7IHRoZW4KICAgICAgICAgICAgICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0ke0hUVFBfUFJPWFlfVVJMf
QogICAgICAgICAgICAgICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBTX1BST1hZX1VSTH0KICAgICAgICAgICAgICAgICAgICBleHBvcnQgSFRUUF9Q
Uk9YWT0ke0hUVFBfUFJPWFlfVVJMfQogICAgICAgICAgICAgICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH0
KICAgICAgICAgICAgICAgICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0Yy9lbnZpcm9ubWVudAogIC
AgICAgICAgICAgICAgICAgIGZpCiAgICAgICAgICAgICAgICAgIGZpCiAgICAgICAgICAgICAgICAgIGZpCiAgICAgICAgICAgICAgICAgIGZpCiAgICAgICAgICAgICAgICAgIENvbmZpZ3VyZSBEb2NrZ
XIgdG8gdXNlIGEgcHJveHkKICAgICAgICAgICAgICAgICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5k
CiAgICAgICAgICAgICAgICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICAgICAgICAgICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJPWFk9JHtIVFRQX1BST1hZX1V
STH1cIgogICAgICAgICAgICAgICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BST1hZPSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgICAgICAgICAgICAgIEVudm
lyb25tZW50PVwiTk9fUFJPWFk9bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9ja
2VyLnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgICAgICAgICAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVsb2FkCiAgICAgICAgICAgICAgICAgIHN5c3RlbWN0
bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgICAgICAgICAgICAgZWNobyAiSW5mbzogZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJ

```
lIG5vdyBjb25maWd1cmVkIHRvIHVzZSB0aGUgcHJveHkgc2V0dGluZ3MiCiAgICB9
  vgpu-license: NVIDIA-client-configuration-token
  nvidia-portal-api-key: API-key-from-NVIDIA-licensing-portal
  password: password-for-vmware-user
```

Hinweis `user-data` ist der base64-codierte Wert für den folgenden cloud-init-Code:

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    set -eu
    source /opt/dlvm/utils.sh
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    set_proxy "http" "https" "socks5"

    DEFAULT_REG_URI="nvcr.io"
    REGISTRY_URI_PATH=$(grep registry-uri /opt/dlvm/ovf-env.xml | sed -n 's/.*oe:value="\
([^"]*\).*/\1/p')

    if [[ -z "$REGISTRY_URI_PATH" ]]; then
      # If REGISTRY_URI_PATH is null or empty, use the default value
      REGISTRY_URI_PATH=$DEFAULT_REG_URI
      echo "REGISTRY_URI_PATH was empty. Using default: $REGISTRY_URI_PATH"
    fi

    # If REGISTRY_URI_PATH contains '/', extract the URI part
    if [[ $REGISTRY_URI_PATH == *"/"* ]]; then
      REGISTRY_URI=$(echo "$REGISTRY_URI_PATH" | cut -d'/' -f1)
    else
      REGISTRY_URI=$REGISTRY_URI_PATH
    fi

    REGISTRY_USERNAME=$(grep registry-user /opt/dlvm/ovf-env.xml | sed -n 's/.*oe:value="\
([^"]*\).*/\1/p')
    REGISTRY_PASSWORD=$(grep registry-passwd /opt/dlvm/ovf-env.xml | sed -n
's/.*oe:value="\([^"]*\).*/\1/p')
    if [[ -n "$REGISTRY_USERNAME" && -n "$REGISTRY_PASSWORD" ]]; then
      docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD $REGISTRY_URI
    else
      echo "Warning: the registry's username and password are invalid, Skipping Docker
login."
    fi

    docker run -d --gpus all -p 8888:8888 $REGISTRY_URI_PATH/nvidia/pytorch:pytorch:23.10-
py3 /usr/local/bin/jupyter lab --allow-root --ip=* --port=8888 --no-browser --
NotebookApp.token='' --NotebookApp.allow_origin='*' --notebook-dir=/workspace

- path: /opt/dlvm/utils.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    error_exit() {
      echo "Error: $1" >&2
      vmtoolsd --cmd "info-set guestinfo.vmservice.bootstrap.condition false,
DLWorkloadFailure, $1"
      exit 1
    }

    check_protocol() {
      local proxy_url=$1
      shift
      local supported_protocols=("$@")
```

```
kind: VirtualMachineService
metadata:
  name: example-dl-vm
  namespace: example-dl-vm-namespace
spec:
  ports:
  - name: ssh
    port: 22
    protocol: TCP
    targetPort: 22
  - name: junyperlab
    port: 8888
    protocol: TCP
    targetPort: 8888
  selector:
    app: example-dl-app
  type: LoadBalancer
```

4 Wechseln Sie zum Kontext des vSphere-Namespace, der vom Cloud-Administrator erstellt wurde.

Beispielsweise für einen Namespace mit der Bezeichnung `example-dl-vm-namespace`:

```
kubectl config use-context example-dl-vm-namespace
```

5 Stellen Sie die Deep Learning-VM bereit.

```
kubectl apply -f example-dl-vm.yaml
```

6 Stellen Sie sicher, dass die VM erstellt wurde, indem Sie diese Befehle ausführen.

```
kubectl get vm -n example-dl-vm-namespace
```

```
kubectl describe virtualmachine example-dl-vm
```

7 Pingen Sie die IP-Adresse der virtuellen Maschine an, die vom angeforderten Netzwerkdienst zugewiesen wurde.

Um die öffentliche Adresse und die Ports für den Zugriff auf die Deep Learning-VM abzurufen, rufen Sie die Details zum Lastausgleichsdienst ab, der erstellt wurde.

```
kubectl get services
```

```
NAME   TYPE           CLUSTER-IP              EXTERNAL-IP
PORT(S)                       AGE
example-dl-vm   LoadBalancer   <internal-ip-address>   <public-IPaddress>   22:30473/
TCP,8888:32180/TCP   9m40s
```

Ergebnisse

Der vGPU-Gasttreiber und die angegebene DL-Arbeitslast werden beim ersten Start der Deep Learning-VM installiert.

Nächste Schritte

- Sie können die Protokolle untersuchen oder das JupyterLab-Notizbuch öffnen, in dem einige der Bilder enthalten sind. Weitere Informationen finden Sie unter Deep Learning-Arbeitslasten in VMware Private AI Foundation with NVIDIA.
- Senden Sie die Zugriffsdetails an Ihre Datenwissenschaftler.

Anpassen der Bereitstellung von Deep Learning-VMs in VMware Private AI Foundation with NVIDIA

Wenn Sie eine Deep Learning-VM in vSphere IaaS control plane mithilfe von `kubectl` oder direkt auf einem vSphere-Cluster bereitstellen, müssen Sie benutzerdefinierte VM-Eigenschaften eingeben.

Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA finden Sie unter Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA.

OVF-Eigenschaften von Deep Learning-VMs

Wenn Sie eine Deep Learning-VM bereitstellen, müssen Sie benutzerdefinierte VM-Eigenschaften ausfüllen, um die Konfiguration des Linux-Betriebssystems, die Bereitstellung des vGPU-Gasttreibers und die Bereitstellung und Konfiguration von NGC-Containern für die DL-Arbeitslasten zu automatisieren.

Das aktuelle Deep Learning-VM-Image verfügt über die folgenden OVF-Eigenschaften:

Kategorie	Parameter	Bezeichnung im vSphere Client	Beschreibung
Eigenschaften des Basisbetriebssystems	instance-id	Instanz-ID	Erforderlich. Eine eindeutige Instanz-ID für die VM-Instanz. Eine Instanz-ID identifiziert eine Instanz eindeutig. Wenn sich eine Instanz-ID ändert, behandelt cloud-init die Instanz als neue Instanz und führt den cloud-init-Prozess erneut für aus.
	hostname	Hostname	Erforderlich. Der Hostname der Appliance.
	seedfrom	URL zum Speichern von Instanzdaten aus	Optional. Eine URL zum Abrufen des Werts für den Parameter „user-data“ und der Metadaten.
	public-keys	Öffentlicher SSH-Schlüssel	Wenn angegeben, füllt die Instanz die SSH-`authorized_keys` des Standardbenutzers mit diesem Wert auf.

Kategorie	Parameter	Bezeichnung im vSphere Client	Beschreibung
	user-data	Codierte Benutzerdaten	Ein Satz von Skripts oder anderen Metadaten, die zum Zeitpunkt der Bereitstellung in die VM eingefügt werden. Diese Eigenschaft stellt den tatsächlichen Inhalt des `cloud-init`-Skripts dar. Dieser Wert muss base64-verschlüsselt werden. ■ Sie können diese Eigenschaft verwenden, um den DL-Arbeitslastcontainer anzugeben, den Sie bereitstellen möchten, z. B. PyTorch oder TensorFlow. Weitere Informationen finden Sie unter Deep Learning-Arbeitslasten in VMware Private AI Foundation with NVIDIA. ■ Sie verwenden diese Eigenschaft, um eine statische IP-Adresse auf eine virtuelle Maschine festzulegen, die direkt auf einem vSphere-Cluster bereitgestellt wird. Weitere Informationen finden Sie unter Zuweisen einer statischen IP-Adresse zu einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA.
	password	Standardbenutzerkennwort	Erforderlich. Das Kennwort für das **vmware**-Standardbenutzerkonto.
vGPU-Treiberinstallation	vgpu-license	vGPU-Lizenz	Erforderlich. Das Konfigurationstoken des NVIDIA vGPU-Clients. Das Token wird in der Datei `/etc/nvidia/ClientConfigToken/client_configuration_token.tok` gespeichert.
	nvidia-portal-api-key	API-Schlüssel des NVIDIA-Portals	In einer verbundenen Umgebung erforderlich. Der API-Schlüssel, den Sie vom NVIDIA-Lizenzierungsportal heruntergeladen haben. Der Schlüssel ist für die Installation des vGPU-Gasttreibers erforderlich.
	vgpu-fallback-version	Version des vGPU-Hosttreibers	Installieren Sie direkt diese Version des vGPU-Gasttreibers.
	vgpu-url	URL für Air-Gap-vGPU-Downloads	In einer nicht verbundenen Umgebung erforderlich. Die URL zum Herunterladen des vGPU-Gasttreibers. Informationen zur notwendigen Konfiguration des lokalen Web-Servers finden Sie unter Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten.

Kategorie	Parameter	Bezeichnung im vSphere Client	Beschreibung
Automatisierung von DL-Arbeitslasten	registry-uri	Registrierungs-URI	Erforderlich in einer nicht verbundenen Umgebung oder wenn Sie eine private Containerregistrierung verwenden möchten, um das Herunterladen von Images aus dem Internet zu vermeiden. Der URI einer privaten Containerregistrierung mit den Deep Learning-Arbeitslast-Container-Images. Erforderlich, wenn Sie auf eine private Registrierung in `user-data` oder `image-oneliner` verweisen.
	registry-user	Registrierungsbenutzername	Erforderlich, wenn Sie eine private Containerregistrierung verwenden, für die eine Standardauthentifizierung erforderlich ist.
	registry-passwd	Registrierungskennwort	Erforderlich, wenn Sie eine private Containerregistrierung verwenden, für die eine Standardauthentifizierung erforderlich ist.
	registry-2-uri	Sekundärer Registrierungs-URI	Erforderlich, wenn Sie eine zweite private Containerregistrierung verwenden, die auf Docker basiert und Standardauthentifizierung erfordert. Wenn Sie beispielsweise eine Deep Learning-VM mit der vorinstallierten NVIDIA RAG-DL-Arbeitslast bereitstellen, wird ein pgvector-Image aus dem Docker-Hub heruntergeladen. Sie können die Parameter `registry-2-` verwenden, um eine Begrenzung der Pull-Rate für docker.io zu umgehen.
	registry-2-user	Sekundärer Registrierungsbenutzername	Erforderlich, wenn Sie eine zweite private Containerregistrierung verwenden.
	registry-2-passwd	Sekundäres Registrierungskennwort	Erforderlich, wenn Sie eine zweite private Containerregistrierung verwenden.
	image-oneliner	Codierter einzeiliger Befehl	Ein einzeilige Bash-Befehl, der bei der VM-Bereitstellung ausgeführt wird. Dieser Wert muss base64-verschlüsselt werden. Sie können diese Eigenschaft verwenden, um den DL-Arbeitslastcontainer anzugeben, den Sie bereitstellen möchten, z. B. PyTorch oder TensorFlow. Weitere Informationen finden Sie unter Deep Learning-Arbeitslasten in VMware Private AI Foundation with NVIDIA. **Vorsicht** Vermeiden Sie die Verwendung von `user-data` und `image-oneliner`.

Kategorie	Parameter	Bezeichnung im vSphere Client	Beschreibung
	docker-compose-uri	Codierte Docker-Erstellungsdatei	Erforderlich, wenn Sie eine Docker-Erstellungsdatei benötigen, um den DL-Arbeitslastcontainer zu starten. Der Inhalt der Datei `docker-compose.yaml`, die bei der Bereitstellung in die virtuelle Maschine eingefügt wird, nachdem die virtuelle Maschine mit aktivierter GPU gestartet wurde. Dieser Wert muss base64-verschlüsselt werden.
	config-json	Codierte config.json	Der Inhalt einer Konfigurationsdatei zum Hinzufügen von Details für Proxyserver. Dieser Wert muss base64-verschlüsselt werden. Weitere Informationen finden Sie unter Bereitstellen einer Deep Learning-VM mit einer Proxyserver.
	conda-environment-install	Installation der Conda-Umgebung	Eine kommagetrennte Liste der Conda-Umgebungen, die nach Abschluss der VM-Bereitstellung automatisch installiert werden sollen. Verfügbare Umgebungen: `pytorch2.3_py3.12`

Deep Learning-Arbeitslasten in VMware Private AI Foundation with NVIDIA

Sie können eine Deep Learning-VM zusätzlich zu ihren eingebetteten Komponenten mit einer unterstützten Deep Learning (DL)-Arbeitslast bereitstellen. Die DL-Arbeitslasten werden aus dem NVIDIA NGC-Katalog heruntergeladen und sind GPU-optimiert und von NVIDIA und VMware von Broadcom validiert.

Eine Übersicht über die Deep Learning-VM-Images finden Sie unter Informationen zu Deep Learning-VM-Images in VMware Private AI Foundation with NVIDIA.

CUDA-Beispiel

Sie können eine Deep Learning-VM mit ausgeführten CUDA-Beispielen verwenden, um die Vektorhinzufügung, die Gravitations-N-Körper-Simulation oder andere Beispiele auf einer VM zu untersuchen. Weitere Informationen finden Sie auf der Seite CUDA-Beispiele.

Nachdem die Deep Learning-VM gestartet wurde, führt sie eine CUDA-Beispielarbeitslast aus, um den vGPU-Gasttreiber zu testen. Sie können die Testausgabe in der Datei `/var/log/dl.log` überprüfen.

Tabelle 3-1. CUDA-Beispiel-Container-Image

Komponente	Beschreibung
Container-Image	`nvcr.io/nvidia/k8s/cuda-sample:ngc_image_tag` Beispiel: `nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8` Informationen zu den CUDA-Beispiel-Container-Images, die für Deep Learning-VMs unterstützt werden, finden Sie unter Versionshinweise zu VMware Deep Learning VM.
Erforderliche Eingaben	Um eine CUDA-Beispielarbeitslast bereitzustellen, müssen Sie die OVF-Eigenschaften für die Deep Learning-VM wie folgt festlegen: ■ Verwenden Sie eine der folgenden Eigenschaften, die für das CUDA-Beispiel-Image spezifisch sind. ■ Cloud-init-Skript. Codieren Sie es im base64-Format.

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    set -eu
    source /opt/dlvm/utils.sh
    set_proxy "http" "https" "socks5"
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    DEFAULT_REG_URI="nvcr.io"
    REGISTRY_URI_PATH=$(grep registry-uri /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')

    if [[ -z "$REGISTRY_URI_PATH" ]]; then
      # If REGISTRY_URI_PATH is null or empty, use the default value
      REGISTRY_URI_PATH=$DEFAULT_REG_URI
      echo "REGISTRY_URI_PATH was empty. Using default:
$REGISTRY_URI_PATH"
    fi

    # If REGISTRY_URI_PATH contains '/', extract the URI part
    if [[ $REGISTRY_URI_PATH == *"/"* ]]; then
      REGISTRY_URI=$(echo "$REGISTRY_URI_PATH" | cut -d'/' -f1)
    else
      REGISTRY_URI=$REGISTRY_URI_PATH
    fi

    REGISTRY_USERNAME=$(grep registry-user /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    REGISTRY_PASSWORD=$(grep registry-passwd /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    if [[ -n "$REGISTRY_USERNAME" && -n "$REGISTRY_PASSWORD" ]];
then
      docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
$REGISTRY_URI
    else
      echo "Warning: the registry's username and password are
invalid, Skipping Docker login."
    fi

    docker run -d $REGISTRY_URI_PATH/nvidia/k8s/cuda-
sample:ngc_image_tag
```

Tabelle 3-1. CUDA-Beispiel-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
- path: /opt/dlvm/utils.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    error_exit() {
      echo "Error: $1" >&2
      vmtoolsd --cmd "info-set
guestinfo.vmservice.bootstrap.condition false, DLWorkloadFailure,
$1"
      exit 1
    }

    check_protocol() {
      local proxy_url=$1
      shift
      local supported_protocols=("$@")
      if [[ -n "${proxy_url}" ]]; then
        local protocol=$(echo "${proxy_url}" | awk -F '://' '{if
(NF > 1) print $1; else print ""}')
        if [ -z "$protocol" ]; then
          echo "No specific protocol provided. Skipping protocol
check."
          return 0
        fi
        local protocol_included=false
        for var in "${supported_protocols[@]}"; do
          if [[ "${protocol}" == "${var}" ]]; then
            protocol_included=true
            break
          fi
        done
        if [[ "${protocol_included}" == false ]]; then
          error_exit "Unsupported protocol: ${protocol}. Supported
protocols are: ${supported_protocols[*]}"
        fi
      fi
    }

    # $@: list of supported protocols
    set_proxy() {
      local supported_protocols=("$@")

      CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
      CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} | base64 --decode)

      HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.http_proxy // empty')
      HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.https_proxy // empty')
      if [[ $? -ne 0 || (-z "${HTTP_PROXY_URL}" && -z "$
{HTTPS_PROXY_URL}") ]]; then
        echo "Info: The config-json was parsed, but no proxy
settings were found."
        return 0
      fi

      check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}"
      check_protocol "${HTTPS_PROXY_URL}" "$
{supported_protocols[@]}"
```

Tabelle 3-1. CUDA-Beispiel-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
      if ! grep -q 'http_proxy' /etc/environment; then
        echo "export http_proxy=${HTTP_PROXY_URL}
        export https_proxy=${HTTPS_PROXY_URL}
        export HTTP_PROXY=${HTTP_PROXY_URL}
        export HTTPS_PROXY=${HTTPS_PROXY_URL}
        export no_proxy=localhost,127.0.0.1" >> /etc/environment
        source /etc/environment
      fi

      # Configure Docker to use a proxy
      mkdir -p /etc/systemd/system/docker.service.d
      echo "[Service]
      Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\"
      Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\"
      Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/
system/docker.service.d/proxy.conf
      systemctl daemon-reload
      systemctl restart docker

      echo "Info: docker and system environment are now configured
to use the proxy settings"
    }
```

Geben Sie beispielsweise für „vectoradd-cuda11.7.1-ubi8“ das folgende Skript im base64-Format an:

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICBz
ZXRfcHJveHkgImh0dHAiICJodHRwcyIgInNvY2tzNSIKICAgIHRyYXAgJ2Vycm9yX2V4
aXQgIlVuZXhwZWN0ZWQgZXJyb3Igb2NjdXJzIGF0IGRsIHdvcmtsb2FkIicgRVJSCiAg
ICBERUZBVUxUX1JFR19VUkk9Im52Y3IuaW8iCiAgICBSRUdJU1RSWV9VUklfUEFUSD0k
KGdyZXAgcmVnaXN0cnktdXJpIC9vcHQvZGx2bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAn
cy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKICAgICBpZiBbWyAteiAiJFJF
R0lTVFJZX1VSSV9QQVRIIiBdXTsgdGhlbgogICAgICAjIElmIFJFR0lTVFJZX1VSSV9Q
QVRIIGlzIG51bGwgb3IgZW1wdHksIHVzZSB0aGUgZGVmYXVsdCB2YWx1ZQogICAgICBS
RUdJU1RSWV9VUklfUEFUSD0kREVGQVVMVF9SRUdfVVJJCiAgICAgICAgIGVjaG8gIlJFR0lT
VFJZX1VSSV9QQVRIIHdhcyBlbXB0eS4gVXNpbmcgZGVmYXVsdDogJFJFR0lTVFJZX1VS
SV9QQVRIIgogICAgZmkKICAgIAogICAgIyBJZiBSRUdJU1RSWV9VUklfUEFUSCBjb250
YWlucyAnLycsIGV4dHJhY3QgdGhlIFVSSSBwYXJ0CiAgICBpZiBbWyAkUkVHSVNUUllf
VVJJX1BBVEggPT0gKiIvIiogXV07IHRoZW4KICAgICAgUkVHSVNUUllfVVJJPSQoZWNo
byAiJFJFR0lTVFJZX1VSSV9QQVRIIiB8IGN1dCAtZCcvJyAtZjEpCiAgICBlbHNlCiAg
ICAgIFJFR0lTVFJZX1VSST0kUkVHSVNUUllfVVJJX1BBVEgKICAgIGZpCiAgICBS
RUdJU1RSWV9VU0VSTkFNRT0kKGdyZXAgcmVnaXN0cnktdXNlciAvb3B0L2Rsdm0vb3Zm
LWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcp
CiAgICBSRUdJU1RSWV9QQVNTV09SRD0kKGdyZXAgcmVnaXN0cnktcGFzc3dkIC9vcHQv
ZGx2bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCku
Ki9cMS9wJykKICAgIGlmIFtbIC1uICIkUkVHSVNUUllfVVNFUk5BTUUiICYmIC1uICIk
UkVHSVNUUllfUEFTU1dPUkQiIF1dOyB0aGVuCiAgICAgIGRvY2tlciBsb2dpbiAtdSAk
UkVHSVNUUllfVVNFUk5BTUUgLXAgJFJFR0lTVFJZX1BBU1NXT1JEICRSRUdJU1RSWV9V
UkkKICAgIGVsc2UKICAgICAgZWNobyAiV2FybmluZzogdGhlIHJlZ2lzdHJ5J3MgdXNl
cm5hbWUgYW5kIHBhc3N3b3JkIGFyZSBpbnZhbGlkLCBTa2lwcGluZyBEb2NrZXIgbG9n
aW4uIgogICAgZmkKICAgIAogICAgZG9ja2VyIHJ1biAtZCAkUkVHSVNUUllfVVJJX1BB
VEgvbnZpZGlhL2s4cy9jdWRhLXNhbXBsZTp2ZWN0b3JhZGQtY3VkYTExLjcuMS11Ymk4
CgotIHBhdGg6IC9vcHQvZGx2bS91dGlscy5zaAogIHBlcm1pc3Npb25zOiAnMDc1NScK
ICBjb250ZW50OiB8CiAgICAjIS9iaW4vYmFzaAogICAgZXJyb3JfZXhpdCgpIHsKICAg
ICAgZWNobyAiRXJyb3I6ICQxIiA+JjIKICAgICAgdm10b29sc2QgLS1jbWQgImluZm8t
c2V0IGd1ZXN0aW5mby52bXNlcnZpY2UuYm9vdHN0cmFwLmNvbmRpdGlvbiBmYWxzZSwg
REExXb3JrbG9hZEZhaWx1cmUsICQxIgogICAgICBleGl0IDEKICAgICAgICBlbHNlCiAg
ICAgICBjaGVj
a19wcm90b2NvbCgpIHsKICAgICAgbG9jYWwgcHJveHlfdXJsPSQxCiAgICAgIHNoaWZ0
```

Tabelle 3-1. CUDA-Beispiel-Container-Image (Fortsetzung)

Komponente	Beschreibung
	CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIpCiAgICAgIGlmIFtb IC1uICIke3Byb3h5X3VybH0iIF1dOyB0aGVuCiAgICAgICAgbG9jYWwgcHJvdG9jb2w9 JChlY2hvICIke3Byb3h5X3VybH0iIHwgYXdrIC1GICc6Ly8nICd7aWYgKE5GID4gMSkg cHJpbnQgJDE7IGVsc2UgcHJpbnQgIiJ9JykKICAgICAgICBpZiBbIC16ICIkcHJvdG9j b2wiIF07IHRoZW4KICAgICAgICAgIGVjaG8gIk5vIHNwZWNpZmljIHByb3RvY29sIHBy b3ZpZGVkLiBTa2lwcGluZyBwcm90b2NvbCBjaGVjay4iCiAgICAgICAgICAgICAgICAg MAogICAgICAgIGZpCiAgICAgICAgbG9jYWwgcHJvdG9jb2xfaW5jbHVkZWQ9ZmFsc2UK ICAgICAgICBmb3IgdmFyIGluICIke3N1cHBvcnRlZF9wcm90b2NvbHNbQF19IjsgZG8K ICAgICAgICAgIGlmIFtbICIke3Byb3RvY29sfSIgPT0gIiR7dmFyfSIgXV07IHRoZW4K ICAgICAgICAgICAgcHJvdG9jb2xfaW5jbHVkZWQ9dHJ1ZQogICAgICAgICAgICBicmVh awogICAgICAgICAgZmkKICAgICAgICBkb25lCiAgICAgICAgaWYgW1sgIiR7cHJvdG9j b2xfaW5jbHVkZWR9IiA9PSBmYWxzZSBdXTsgdGhlbgogICAgICAgICAgZXJyb3JfZXhp dCAiVW5zdXBwb3J0ZWQgcHJvdG9jb2w6ICR7cHJvdG9jb2x9LiBTdXBwb3J0ZWQgcHJv dG9jb2xzIGFyZTogJHtzdXBwb3J0ZWRfcHJvdG9jb2xzWypdfSIKICAgICAgICBmaQog ICAgICBmaQogICAgfQoKICAgICMgJEA6IGxpc3Qgb2Ygc3VwcG9ydGVkIHByb3RvY29s cwogICAgc2V0X3Byb3h5KCkgewogICAgICBsb2NhbCBzdXBwb3J0ZWRfcHJvdG9jb2xz PSgiJEAiKQoKICAgICAgQ09ORklHX0pTT05fQkFTRTY0PSQoZ3JlcCAnY29uZmlnLWpz b24nIC9vcHQvZGx2bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJc KFteIl0qXCkuKi9cMS9wJykKICAgICAgQ09ORklHX0pTT049JChlY2hvICR7Q09ORklH X0pTT05fQkFTRTY0fSB8IGJhc2U2NCAtLWRlY29kZSkKCiAgICAgICAgIEhUVFBfUFJPWFlf VVJMPSQoZWNobyAiJHtDT05GSUdfSlNPTn0iIHwganEgLXIgJy5odHRwX3Byb3h5IC8v IGVtcHR5JykKICAgICAgSFRUUFNfUFJPWFlfVVJMPSQoZWNobyAiJHtDT05GSUdfSlNP Tn0iIHwganEgLXIgJy5odHRwc19wcm94eSAvLyBlbXB0eScpCiAgICAgIGlmIFtbICQ/ IC1uZSAwIHx8ICgteiAiJHtIVFRQX1BST1hZX1VSTH0iICYmIC16ICIke0hUVFBTX1BS T1hZX1VSTH0iKSBdXTsgdGhlbgogICAgICAgIGVjaG8gIkluZm86IFRoZSBjb25maWct anNvbiB3YXMgcGFyc2VkLCBidXQgbm8gcHJveHkgc2V0dGluZ3Mgd2VyZSBmb3VuZC4i CiAgICAgICAgcmV0dXJuIDAKICAgICAgZmkKICAKICAgICAgY2hlY2tfcHJvdG9jb2wg IiR7SFRUUF9QUk9YWV9VUkx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAg ICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVk X3Byb3RvY29sc1tAXX0iCgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9l dGMvZW52aXJvbm1lbnQ7IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94 eT0ke0hUVFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hU VFBTX1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJP WFlfVVJMfQogICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VS TH0KICAgICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4g L2V0Yy9lbnZpcm9ubWVudAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAg ICAgIGZpCiAgICAgIAogICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJv eHkKICAgICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2Vydmlj ZS5kCiAgICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBf UFJPWFk9JHtIVFRQX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBT X1BST1hZPSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25tZW50PVwiTk9f UFJPWFk9bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0v ZG9ja2VyLnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgICAgIHN5c3RlbWN0bCByZXN0 cmVsb2FkCiAgICAgICAgIHN5c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAi SW5mbzogZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1 cmVkIHRvIHVzZSB0aGUgcHJveHkgc2V0dGluZ3MiCiAgICB9

was dem folgenden Skript im Klartextformat entspricht:

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    set -eu
    source /opt/dlvm/utils.sh
    set_proxy "http" "https" "socks5"
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    DEFAULT_REG_URI="nvcr.io"
```

Tabelle 3-1. CUDA-Beispiel-Container-Image (Fortsetzung)

Komponente	Beschreibung
	REGISTRY_URI_PATH=$(grep registry-uri /opt/dlvm/ovf-env.xml \| sed -n 's/.*oe:value="\([^"]*\).*/\1/p') if [[-z "$REGISTRY_URI_PATH"]]; then # If REGISTRY_URI_PATH is null or empty, use the default value REGISTRY_URI_PATH=$DEFAULT_REG_URI echo "REGISTRY_URI_PATH was empty. Using default: $REGISTRY_URI_PATH" fi # If REGISTRY_URI_PATH contains '/', extract the URI part if [[$REGISTRY_URI_PATH == *"/"*]]; then REGISTRY_URI=$(echo "$REGISTRY_URI_PATH" \| cut -d'/' -f1) else REGISTRY_URI=$REGISTRY_URI_PATH fi REGISTRY_USERNAME=$(grep registry-user /opt/dlvm/ovf-env.xml \| sed -n 's/.*oe:value="\([^"]*\).*/\1/p') REGISTRY_PASSWORD=$(grep registry-passwd /opt/dlvm/ovf-env.xml | sed -n 's/.*oe:value="\([^"]*\).*/\1/p') if [[-n "$REGISTRY_USERNAME" && -n "$REGISTRY_PASSWORD"]]; then docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD $REGISTRY_URI else echo "Warning: the registry's username and password are invalid, Skipping Docker login." fi docker run -d $REGISTRY_URI_PATH/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8 - path: /opt/dlvm/utils.sh permissions: '0755' content: | #!/bin/bash error_exit() { echo "Error: $1" >&2 vmtoolsd --cmd "info-set guestinfo.vmservice.bootstrap.condition false, DLWorkloadFailure, $1" exit 1 } check_protocol() { local proxy_url=$1 shift local supported_protocols=("$@") if [[-n "${proxy_url}"]]; then local protocol=$(echo "${proxy_url}" | awk -F '://' '{if (NF > 1) print $1; else print ""}') if [-z "$protocol"]; then echo "No specific protocol provided. Skipping protocol check." return 0 fi local protocol_included=false for var in "${supported_protocols[@]}"; do if [["${protocol}" == "${var}"]]; then protocol_included=true

Tabelle 3-1. CUDA-Beispiel-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
            break
          fi
        done
        if [[ "${protocol_included}" == false ]]; then
          error_exit "Unsupported protocol: ${protocol}. Supported
protocols are: ${supported_protocols[*]}"
        fi
      fi
    }

    # $@: list of supported protocols
    set_proxy() {
      local supported_protocols=("$@")

      CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
      CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} | base64 --decode)

      HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.http_proxy // empty')
      HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.https_proxy // empty')
      if [[ $? -ne 0 || (-z "${HTTP_PROXY_URL}" && -z "$
{HTTPS_PROXY_URL}") ]]; then
        echo "Info: The config-json was parsed, but no proxy
settings were found."
        return 0
      fi

      check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}"
      check_protocol "${HTTPS_PROXY_URL}" "$
{supported_protocols[@]}"

      if ! grep -q 'http_proxy' /etc/environment; then
        echo "export http_proxy=${HTTP_PROXY_URL}
        export https_proxy=${HTTPS_PROXY_URL}
        export HTTP_PROXY=${HTTP_PROXY_URL}
        export HTTPS_PROXY=${HTTPS_PROXY_URL}
        export no_proxy=localhost,127.0.0.1" >> /etc/environment
        source /etc/environment
      fi

      # Configure Docker to use a proxy
      mkdir -p /etc/systemd/system/docker.service.d
      echo "[Service]
      Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\"
      Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\"
      Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/
system/docker.service.d/proxy.conf
      systemctl daemon-reload
      systemctl restart docker

      echo "Info: docker and system environment are now configured
to use the proxy settings"
    }
```

- Einzeiliges Image. Im base64-Format codieren

```
docker run -d nvcr.io/nvidia/k8s/cuda-sample:ngc_image_tag
```

Tabelle 3-1. CUDA-Beispiel-Container-Image (Fortsetzung)

Komponente	Beschreibung
	Geben Sie beispielsweise für „vectoradd-cuda11.7.1-ubi8“ das folgende Skript im base64-Format an: `ZG9ja2VyIHJ1biAtZCBudmNyLmlvL252aWRpYS9rOHMvY3VkYS1zYW1wbGU6dmVjdG9y YWRkLWN1ZGExMS43LjEtdWJpOA==` was dem folgenden Skript im Klartextformat entspricht: `docker run -d nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8` ■ Geben Sie die Installationseigenschaften des vGPU-Gasttreibers ein, wie z. B. `vgpu-license` und `nvidia-portal-api-key`. ■ Geben Sie nach Bedarf Werte für die Eigenschaften an, die für eine getrennte Umgebung erforderlich sind. Weitere Informationen finden Sie unter OVF-Eigenschaften von Deep Learning-VMs.
Ausgabe	■ Installationsprotokolle für den vGPU-Gasttreiber in `/var/log/vgpu-install.log`. Führen Sie den folgenden Befehl aus, um sicherzustellen, dass der vGPU-Gasttreiber installiert und die Lizenz zugeteilt ist: `nvidia-smi -q |grep -i license` ■ Cloud-init-Skriptprotokolle in `/var/log/dl.log`.

PyTorch

Sie können eine Deep Learning-VM mit einer PyTorch-Bibliothek verwenden, um Konversations-KI, NLP und andere Arten von KI-Modellen auf einer VM zu erkunden. Weitere Informationen finden Sie auf der Seite PyTorch.

Nach dem Start der Deep Learning-VM wird eine JupyterLab-Instanz mit installierten und konfigurierten PyTorch-Paketen gestartet.

Tabelle 3-2. PyTorch-Container-Image

Komponente	Beschreibung
Container-Image	`nvcr.io/nvidia/pytorch:ngc_image_tag` Beispiel: `nvcr.io/nvidia/pytorch:23.10-py3` Informationen zu den PyTorch-Container-Images, die für Deep Learning-VMs unterstützt werden, finden Sie unter Versionshinweise zu VMware Deep Learning VM.
Erforderliche Eingaben	Um eine PyTorch-Arbeitslast bereitzustellen, müssen Sie die OVF-Eigenschaften für die Deep Learning-VM wie folgt festlegen: ■ Verwenden Sie eine der folgenden Eigenschaften, die für das PyTorch-Image spezifisch sind. ■ Cloud-init-Skript. Codieren Sie es im base64-Format. `#cloud-config` `write_files:` `- path: /opt/dlvm/dl_app.sh` `permissions: '0755'` `content: |` `#!/bin/bash` `set -eu` `source /opt/dlvm/utils.sh` `trap 'error_exit "Unexpected error occurs at dl workload"' ERR` `set_proxy "http" "https" "socks5"` `DEFAULT_REG_URI="nvcr.io"` `REGISTRY_URI_PATH=$(grep registry-uri /opt/dlvm/ovf-env.xml \| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')` `if [[ -z "$REGISTRY_URI_PATH" ]]; then` `# If REGISTRY_URI_PATH is null or empty, use the default value` `REGISTRY_URI_PATH=$DEFAULT_REG_URI` `echo "REGISTRY_URI_PATH was empty. Using default: $REGISTRY_URI_PATH"` `fi` `# If REGISTRY_URI_PATH contains '/', extract the URI part` `if [[ $REGISTRY_URI_PATH == *"/"* ]]; then` `REGISTRY_URI=$(echo "$REGISTRY_URI_PATH" \| cut -d'/' -f1)` `else` `REGISTRY_URI=$REGISTRY_URI_PATH` `fi` `REGISTRY_USERNAME=$(grep registry-user /opt/dlvm/ovf-env.xml \| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')` `REGISTRY_PASSWORD=$(grep registry-passwd /opt/dlvm/ovf-env.xml | sed -n 's/.*oe:value="\([^"]*\).*/\1/p')` `if [[ -n "$REGISTRY_USERNAME" && -n "$REGISTRY_PASSWORD" ]]; then` `docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD $REGISTRY_URI` `else` `echo "Warning: the registry's username and password are invalid, Skipping Docker login."` `fi` `docker run -d --gpus all -p 8888:8888 $REGISTRY_URI_PATH/nvidia/pytorch:ngc_image_tag /usr/local/bin/jupyter lab --allow-`

Tabelle 3-2. PyTorch-Container-Image (Fortsetzung)

Komponente	Beschreibung
	root --ip=* --port=8888 --no-browser --NotebookApp.token='' --NotebookApp.allow_origin='*' --notebook-dir=/workspace - path: /opt/dlvm/utils.sh permissions: '0755' content: | #!/bin/bash error_exit() { echo "Error: $1" >&2 vmtoolsd --cmd "info-set guestinfo.vmservice.bootstrap.condition false, DLWorkloadFailure, $1" exit 1 } check_protocol() { local proxy_url=$1 shift local supported_protocols=("$@") if [[-n "${proxy_url}"]]; then local protocol=$(echo "${proxy_url}" | awk -F '://' '{if (NF > 1) print $1; else print ""}') if [-z "$protocol"]; then echo "No specific protocol provided. Skipping protocol check." return 0 fi local protocol_included=false for var in "${supported_protocols[@]}"; do if [["${protocol}" == "${var}"]]; then protocol_included=true break fi done if [["${protocol_included}" == false]]; then error_exit "Unsupported protocol: ${protocol}. Supported protocols are: ${supported_protocols[*]}" fi fi } # $@: list of supported protocols set_proxy() { local supported_protocols=("$@") CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml \| sed -n 's/.*oe:value="\([^"]*\).*/\1/p') CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} \| base64 --decode) HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" \| jq -r '.http_proxy // empty') HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r '.https_proxy // empty') if [[$? -ne 0 \|\| (-z "${HTTP_PROXY_URL}" && -z "${HTTPS_PROXY_URL}")]]; then echo "Info: The config-json was parsed, but no proxy settings were found." return 0 fi check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}"

Tabelle 3-2. PyTorch-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
      check_protocol "${HTTPS_PROXY_URL}" "$
{supported_protocols[@]}"

      if ! grep -q 'http_proxy' /etc/environment; then
        echo "export http_proxy=${HTTP_PROXY_URL}
        export https_proxy=${HTTPS_PROXY_URL}
        export HTTP_PROXY=${HTTP_PROXY_URL}
        export HTTPS_PROXY=${HTTPS_PROXY_URL}
        export no_proxy=localhost,127.0.0.1" >> /etc/environment
        source /etc/environment
      fi

      # Configure Docker to use a proxy
      mkdir -p /etc/systemd/system/docker.service.d
      echo "[Service]
      Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\"
      Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\"
      Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/
system/docker.service.d/proxy.conf
      systemctl daemon-reload
      systemctl restart docker

      echo "Info: docker and system environment are now configured
to use the proxy settings"
    }
```

Geben Sie beispielsweise für „pytorch:23.10-py3“ das folgende Skript im base64-Format an:

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCgog
ICAgREVGQVVMVF9SRUdfVVJJPSJudmNyLmlvIgogICAgUkVHSVNUUllfVVJJX1BBVEg9
JChncmVwIHJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4g
J3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCgogICAgaWYgW1sgLXogIiRS
RUdJU1RSWV9VUklfUEFUSCIgXV07IHRoZW4KICAgICAgIyBJZiBSRUdJU1RSWV9VUklf
UEFUSCBpcyBudWxsIG9yIGVtcHR5LCB1c2UgdGhlIGRlZmF1bHQgdmFsdWUKICAgICAg
UkVHSVNUUllfVVJJX1BBVEg9JERFRkFVTFRfUkVHX1VSSQogICAgICBlY2hvICJSRUdJ
U1RSWV9VUklfUEFUSCB3YXMgZW1wdHkuIFVzaW5nIGRlZmF1bHQ6ICRSRUdJU1RSWV9V
UklfUEFUSCIKICAgIGZpCiAgICAKICAgICMgSWYgUkVHSVNUUllfVVJJX1BBVEggY29u
dGFpbnMgJy8nLCBleHRyYWN0IHRoZSBVUkkgcGFydAogICAgaWYgW1sgJFJFR0lTVFJZ
X1VSSV9QQVRIID09ICIoiLyIqIF1dOyB0aGVuCiAgICAgIFJFR0lTVFJZX1VSST0kKGVj
aG8gIiRSRUdJU1RSWV9VUklfUEFUSCIgfCBjdXQgLWQnLycgLWYxKQogICAgZWxzZQog
ICAgICBSRUdJU1RSWV9VUkk9JFJFR0lTVFJZX1VSSV9QQVRICiAgICBmaQogICAgICAg
UkVHSVNUUllfVVNFUk5BTUU9JChncmVwIHJlZ2lzdHJ5LXVzZXIgL29wdC9kbHZtL292
Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3An
KQogICAgUkVHSVNUUllfUEFTU1dPUkQ9JChncmVwIHJlZ2lzdHJ5LXBhc3N3ZCAvb3B0
L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwp
LiovXDEvcCcpCiAgICBpZiBbWyAtbiAiJFJFR0lTVFJZX1VTRVJOQU1FIiAmJiAtbiAi
JFJFR0lTVFJZX1BBU1NXT1JEIiBdXTsgdGhlbgogICAgICBkb2NrZXIgbG9naW4gLXUg
JFJFR0lTVFJZX1VTRVJOQU1FIC1wICRSRUdJU1RSWV9QQVNTV09SRCAkUkVHSVNUUllf
VVJJCiAgICBlbHNlCiAgICAgIGVjaG8gIldhcm5pbmc6IHRoZSByZWdpc3RyeSdzIHVz
ZXJuYW1lIGFuZCBwYXNzd29yZCBhcmUgaW52YWxpZCwgU2tpcHBpbmcgRG9ja2VyIGxv
Z2luLiIKICAgIGZpCgogICAgZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC1wIDg4ODg6
ODg4OCAkUkVHSVNUUllfVVJJX1BBVEgvbnZpZGlhL3B5dG9yY2g6MjMuMTAtcHkzIC91
c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9vdCAtLWlwPSogLS1wb3J0
PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAudG9rZW49JycgLS1Ob3RlYm9v
a0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGlyPS93b3Jrc3BhY2UKCi0g
```

Tabelle 3-2. PyTorch-Container-Image (Fortsetzung)

Komponente	Beschreibung
	cGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNv bnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAgICBl Y2hvICJFcnJvcjogJDEiID4mMgogICAgICB2bXRvb2xzZCAtLWNtZCAiaW5mby1zZXQg Z3Vlc3RpbmZvLnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIGZhbHNlLCBETFdv cmtsb2FkRmFpbHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgfQoKICAgIGNoZWNrX3By b3RvY29sKCkgewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQKICAg ICAgbG9jYWwgc3VwcG9ydGVkX3Byb3RvY29scy0oIiRAIikKICAgICAgaWYgW1sgLW4g IiR7cHJveHlfdXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm90b2NvbD0kKGVj aG8iR7cHJveHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBwcmlu dCAkMTsgZWxzZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2NvbCIg XTsgdGhlbgogICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMgcHJvdG9jb2wgcHJvdmlk ZWQuIFNraXBwaW5nIHByb3RvY29sIGNoZWNrLiIKICAgICAgICAgIHJldHVybiAwCiAg ICAgICAgZmkKICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQogICAg ICAgIGZvciB2YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwogICAg ICAgICAgaWYgW1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9IiBdXTsgdGhlbgogICAg ICAgICAgICBwcm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgIGJyZWFrCiAg ICAgICAgZmkKICAgICAgICBmaQogICAgICAgIGRvbmUKICAgICAgICBpZiBbIWyAiJHtwcm90b2NvbF9p bmNsdWRlZH0iID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0ICJV bnN1cHBvcnRlZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1cHBvcnRlZCBwcm90b2Nv bHMgYXJlOiAke3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgICAgZXZpCiAg ICAgICAgICB9CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xzCiAg ICBzZXRfcHJveHkoKSB7CiAgICAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIk QCIpCgogICAgICBDT05GSUdfSlNPTl9CQVNFNjQ9JChncmVwICdjb25maWctanNvbicg L29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14i XSpcKS4qL1wxL3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdfSlNP Tl9CQVNFNjR9IHwgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgSFRUUF9QUk9YWV9VUkw9 JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0dHBfcHJveHkgLy8gZW1w dHknKQogICAgICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09OfSIg fCBqcSAtciAnLmh0dHBzX3Byb3h5IC8vIGVtcHR5JykKICAgICAgaWYgW1sgJD8gLW5l IDAgfHwgKC16ICIke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJPWFlf VVJMfSIpIF1dOyB0aGVuCiAgICAgICAgZWNobyAiSW5mbzogVGhlIGNvbmZpZy1qc29u IHdhcyBwYXJzZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kLiIKICAg ICAgICByZXR1cm4gMAogICAgICBmaQoKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRU UF9QUk9YWV9Ukx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAgY2hl Y2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVkX3Byb3Rv Y29sc1tAXX0iCgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldGMvZW52 aXJvbm1lbnQ7IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0ke0hU VFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBTX1BS T1hZX1VSTH0KICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJPWFlfVVJM fQogICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1BST1hZX1VSTH0K ICAgICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0Yy9l bnZpcm9ubWVudAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAgICAgIGZp CiAgICAgIAogICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJveHkKICAg ICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5kCiAg ICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJPWFk9 JHtIVFRQX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BST1hZ PSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgICAgIEVudmlyb25tZW50PVwiTk9fUFJPWFk9 bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9ja2Vy LnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVsb2Fk CiAgICAgICAgIHN5c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAiSW5mbzog ZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1cmVkIHRv IHVzZSB0aGUgcHJveHkgc2V0dGluZ3MiCiAgICB9

was dem folgenden Skript im Klartextformat entspricht:

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
  content: |
```

Tabelle 3-2. PyTorch-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
    #!/bin/bash
    set -eu
    source /opt/dlvm/utils.sh
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    set_proxy "http" "https" "socks5"

    DEFAULT_REG_URI="nvcr.io"
    REGISTRY_URI_PATH=$(grep registry-uri /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')

    if [[ -z "$REGISTRY_URI_PATH" ]]; then
      # If REGISTRY_URI_PATH is null or empty, use the default value
      REGISTRY_URI_PATH=$DEFAULT_REG_URI
      echo "REGISTRY_URI_PATH was empty. Using default:
$REGISTRY_URI_PATH"
    fi

    # If REGISTRY_URI_PATH contains '/', extract the URI part
    if [[ $REGISTRY_URI_PATH == *"/"* ]]; then
      REGISTRY_URI=$(echo "$REGISTRY_URI_PATH" | cut -d'/' -f1)
    else
      REGISTRY_URI=$REGISTRY_URI_PATH
    fi

    REGISTRY_USERNAME=$(grep registry-user /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    REGISTRY_PASSWORD=$(grep registry-passwd /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    if [[ -n "$REGISTRY_USERNAME" && -n "$REGISTRY_PASSWORD" ]];
then
      docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
$REGISTRY_URI
    else
      echo "Warning: the registry's username and password are
invalid, Skipping Docker login."
    fi

    docker run -d --gpus all -p 8888:8888 $REGISTRY_URI_PATH/
nvidia/pytorch:23.10-py3 /usr/local/bin/jupyter lab --allow-root
--ip=* --port=8888 --no-browser --NotebookApp.token='' --
NotebookApp.allow_origin='*' --notebook-dir=/workspace

- path: /opt/dlvm/utils.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    error_exit() {
      echo "Error: $1" >&2
      vmtoolsd --cmd "info-set
guestinfo.vmservice.bootstrap.condition false, DLWorkloadFailure,
$1"
      exit 1
    }

    check_protocol() {
      local proxy_url=$1
      shift
      local supported_protocols=("$@")
      if [[ -n "${proxy_url}" ]]; then
        local protocol=$(echo "${proxy_url}" | awk -F '://' '{if
(NF > 1) print $1; else print ""}')
```

Tabelle 3-2. PyTorch-Container-Image (Fortsetzung)

Komponente	Beschreibung
	<pre> if [-z "$protocol"]; then echo "No specific protocol provided. Skipping protocol check." return 0 fi local protocol_included=false for var in "${supported_protocols[@]}"; do if [["${protocol}" == "${var}"]]; then protocol_included=true break fi done if [["${protocol_included}" == false]]; then error_exit "Unsupported protocol: ${protocol}. Supported protocols are: ${supported_protocols[*]}" fi fi } # $@: list of supported protocols set_proxy() { local supported_protocols=("$@") CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml | sed -n 's/.*oe:value="\([^"]*\).*/\1/p') CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} \| base64 --decode) HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" \| jq -r '.http_proxy // empty') HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r '.https_proxy // empty') if [[$? -ne 0 \|\| (-z "${HTTP_PROXY_URL}" && -z "${HTTPS_PROXY_URL}")]]; then echo "Info: The config-json was parsed, but no proxy settings were found." return 0 fi check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}" check_protocol "${HTTPS_PROXY_URL}" "${supported_protocols[@]}" if ! grep -q 'http_proxy' /etc/environment; then echo "export http_proxy=${HTTP_PROXY_URL} export https_proxy=${HTTPS_PROXY_URL} export HTTP_PROXY=${HTTP_PROXY_URL} export HTTPS_PROXY=${HTTPS_PROXY_URL} export no_proxy=localhost,127.0.0.1" >> /etc/environment source /etc/environment fi # Configure Docker to use a proxy mkdir -p /etc/systemd/system/docker.service.d echo "[Service] Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\" Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\" Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/system/docker.service.d/proxy.conf systemctl daemon-reload systemctl restart docker</pre>

Tabelle 3-2. PyTorch-Container-Image (Fortsetzung)

Komponente	Beschreibung
	<pre> echo "Info: docker and system environment are now configured to use the proxy settings" }</pre> ■ Einzeiliges Image. Codieren Sie es im base64-Format. <pre>docker run -d -p 8888:8888 nvcr.io/nvidia/pytorch:*ngc_image_tag* /usr/local/bin/jupyter lab --allow-root --ip=* --port=8888 --no-browser --NotebookApp.token='' --NotebookApp.allow_origin='*' --notebook-dir=/workspace</pre> Geben Sie beispielsweise für „pytorch:23.10-py3“ das folgende Skript im base64-Format an: <pre>ZG9ja2VyIHJ1biAtZCAtcCA4ODg4Ojg4ODggbnZjci5pby9udmlkaWEvcHl0b3JjaDoyMy4xMC1weTMgL3Vzci9sb2NhbC9iaW4vanVweXRlciBsYWIgLS1hbGxvdy1yb290IC0taXA9KiAtLXBvcnQ9ODg4OCAtLW5vLWJyb3dzZXIgLS1Ob3RlYm9va0FwcC50b2tlbj0nJyAtLU5vdGVib29rQXBwLmFsbG93X29yaWdpbj0nKicgLS1ub3RlYm9vay1kaXI9L3dvcmtzcGFjZQ==</pre> was dem folgenden Skript im Klartextformat entspricht: <pre>docker run -d -p 8888:8888 nvcr.io/nvidia/pytorch:23.10-py3 /usr/local/bin/jupyter lab --allow-root --ip=* --port=8888 --no-browser --NotebookApp.token='' --NotebookApp.allow_origin='*' --notebook-dir=/workspace</pre> ■ Geben Sie die Installationseigenschaften des vGPU-Gasttreibers ein, wie z. B. `vgpu-license` und `nvidia-portal-api-key`. ■ Geben Sie nach Bedarf Werte für die Eigenschaften an, die für eine getrennte Umgebung erforderlich sind. Weitere Informationen finden Sie unter OVF-Eigenschaften von Deep Learning-VMs.
Ausgabe	■ Installationsprotokolle für den vGPU-Gasttreiber in `/var/log/vgpu-install.log`. Um zu überprüfen, ob der vGPU-Gasttreiber installiert ist, führen Sie den Befehl `nvidia-smi` aus. ■ Cloud-init-Skriptprotokolle in `/var/log/dl.log`. ■ PyTorch-Container. Um zu überprüfen, ob der PyTorch-Container ausgeführt wird, führen Sie die Befehle `sudo docker ps -a` und `sudo docker logs *container_id*` aus. ■ JupyterLab-Instanz, auf die Sie unter `http://*dl_vm_ip*:8888` zugreifen können. Stellen Sie im Terminal von JupyterLab sicher, dass die folgenden Funktionen im Notizbuch verfügbar sind: ■ Um zu überprüfen, ob JupyterLab auf die vGPU-Ressource zugreifen kann, führen Sie `nvidia-smi` aus. ■ Um sicherzustellen, dass die PyTorch-bezogenen Pakete installiert sind, führen Sie `pip show` aus.

TensorFlow

Sie können eine Deep Learning-VM mit einer TensorFlow-Bibliothek verwenden, um Konversations-KI, NLP und andere Arten von KI-Modellen auf einer VM zu erkunden. Weitere Informationen finden Sie auf der Seite TensorFlow.

Nachdem die Deep Learning-VM gestartet wurde, startet sie eine JupyterLab-Instanz mit installierten und konfigurierten TensorFlow-Paketen.

Tabelle 3-3. TensorFlow-Container-Image

Komponente	Beschreibung
Container-Image	`nvcr.io/nvidia/tensorflow:ngc_image_tag` Beispiel: `nvcr.io/nvidia/tensorflow:23.10-tf2-py3` Informationen zu den TensorFlow-Container-Images, die für Deep Learning-VMs unterstützt werden, finden Sie unter Versionshinweise zu VMware Deep Learning VM.
Erforderliche Eingaben	Um eine TensorFlow-Arbeitslast bereitzustellen, müssen Sie die OVF-Eigenschaften für die Deep Learning-VM wie folgt festlegen: ■ Verwenden Sie eine der folgenden Eigenschaften, die für das TensorFlow-Image spezifisch sind. ■ Cloud-init-Skript. Codieren Sie es im base64-Format. `#cloud-config` `write_files:` `- path: /opt/dlvm/dl_app.sh` `permissions: '0755'` `content: |` `#!/bin/bash` `set -eu` `source /opt/dlvm/utils.sh` `trap 'error_exit "Unexpected error occurs at dl workload"' ERR` `set_proxy "http" "https" "socks5"` `DEFAULT_REG_URI="nvcr.io"` `REGISTRY_URI_PATH=$(grep registry-uri /opt/dlvm/ovf-env.xml \| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')` `if [[ -z "$REGISTRY_URI_PATH" ]]; then` `# If REGISTRY_URI_PATH is null or empty, use the default value` `REGISTRY_URI_PATH=$DEFAULT_REG_URI` `echo "REGISTRY_URI_PATH was empty. Using default: $REGISTRY_URI_PATH"` `fi` `# If REGISTRY_URI_PATH contains '/', extract the URI part` `if [[ $REGISTRY_URI_PATH == *"/"* ]]; then` `REGISTRY_URI=$(echo "$REGISTRY_URI_PATH" \| cut -d'/' -f1)` `else` `REGISTRY_URI=$REGISTRY_URI_PATH` `fi` `REGISTRY_USERNAME=$(grep registry-user /opt/dlvm/ovf-env.xml \| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')` `REGISTRY_PASSWORD=$(grep registry-passwd /opt/dlvm/ovf-env.xml | sed -n 's/.*oe:value="\([^"]*\).*/\1/p')` `if [[ -n "$REGISTRY_USERNAME" && -n "$REGISTRY_PASSWORD" ]]; then` `docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD $REGISTRY_URI` `else` `echo "Warning: the registry's username and password are invalid, Skipping Docker login."` `fi` `docker run -d --gpus all -p 8888:8888 $REGISTRY_URI_PATH/`

Tabelle 3-3. TensorFlow-Container-Image (Fortsetzung)

Komponente	Beschreibung
	nvidia/tensorflow:*ngc_image_tag* /usr/local/bin/jupyter lab --allow-root --ip=* --port=8888 --no-browser --NotebookApp.token='' --NotebookApp.allow_origin='*' --notebook-dir=/workspace - path: /opt/dlvm/utils.sh permissions: '0755' content: | #!/bin/bash error_exit() { echo "Error: $1" >&2 vmtoolsd --cmd "info-set guestinfo.vmservice.bootstrap.condition false, DLWorkloadFailure, $1" exit 1 } check_protocol() { local proxy_url=$1 shift local supported_protocols=("$@") if [[-n "${proxy_url}"]]; then local protocol=$(echo "${proxy_url}" | awk -F '://' '{if (NF > 1) print $1; else print ""}') if [-z "$protocol"]; then echo "No specific protocol provided. Skipping protocol check." return 0 fi local protocol_included=false for var in "${supported_protocols[@]}"; do if [["${protocol}" == "${var}"]]; then protocol_included=true break fi done if [["${protocol_included}" == false]]; then error_exit "Unsupported protocol: ${protocol}. Supported protocols are: ${supported_protocols[*]}" fi fi } # $@: list of supported protocols set_proxy() { local supported_protocols=("$@") CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml \| sed -n 's/.*oe:value="\([^"]*\).*/\1/p') CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} \| base64 --decode) HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" \| jq -r '.http_proxy // empty') HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r '.https_proxy // empty') if [[$? -ne 0 \|\| (-z "${HTTP_PROXY_URL}" && -z "${HTTPS_PROXY_URL}")]]; then echo "Info: The config-json was parsed, but no proxy settings were found." return 0 fi

Tabelle 3-3. TensorFlow-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
      check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}"
      check_protocol "${HTTPS_PROXY_URL}" "$
{supported_protocols[@]}"

      if ! grep -q 'http_proxy' /etc/environment; then
        echo "export http_proxy=${HTTP_PROXY_URL}
        export https_proxy=${HTTPS_PROXY_URL}
        export HTTP_PROXY=${HTTP_PROXY_URL}
        export HTTPS_PROXY=${HTTPS_PROXY_URL}
        export no_proxy=localhost,127.0.0.1" >> /etc/environment
        source /etc/environment
      fi

      # Configure Docker to use a proxy
      mkdir -p /etc/systemd/system/docker.service.d
      echo "[Service]
      Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\"
      Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\"
      Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/
system/docker.service.d/proxy.conf
      systemctl daemon-reload
      systemctl restart docker

      echo "Info: docker and system environment are now configured
to use the proxy settings"
    }
```

Geben Sie beispielsweise für „tensorflow:23.10-tf2-py3“ das folgende Skript im base64-Format an:

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkb3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCiAg
ICAKICAgIERFRkFVTFRfUkVHX1VSST0ibnZjci5pbyIKICAgIFJFR0lTVFJZX1VSSV9Q
QVRIPSQoZ3JlcCByZWdpc3RyeS11cmkgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2Vk
IC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIGlmIFtbIC16
ICIkUkVHSVNUUllfVVJJX1BBVEgiIF1dOyB0aGVuCiAgICAgICMgSWYgUkVHSVNUUllf
VVJJX1BBVEggaXMgbnVsbCBvciBlbXB0eSwgdXNlIHRoZSBkZWZhdWx0IHZhbHVlCiAg
ICAgIFJFR0lTVFJZX1VSSV9QQVRIPSRERUZBVUxUX1JFR19VUkkKICAgICAgZWNobyAi
UkVHSVNUUllfVVJJX1BBVEggd2FzIGVtcHR5LiBVc2luZyBkZWZhdWx0OiAkUkVHSVNU
UllfVVJJX1BBVEgiCiAgICBmaQogICAgICAgICAgICAjIElmIFJFR0lTVFJZX1VSSV9QQVRI
IGNvbnRhaW5zICcvJywgZXh0cmFjdCB0aGUgVVJJIHBhcnQKICAgIGlmIFtbICRSRUdJ
U1RSWV9VUklfUEFUSCA9PSAqIi8iKiBdXTsgdGhlbgogICAgICBSRUdJU1RSWV9VUkk9
JChlY2hvICIkUkVHSVNUUllfVVJJX1BBVEgiIHwgY3V0IC1kJy8nIC1mMSkKICAgIGVs
c2UKICAgICAgUkVHSVNUUllfVVJJX1BBVEg9JFSQoZ3JlcCByZWdpc3RyeS11c2VyIC9vcHQvZGx2
bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9c
MS9wJykKICAgIFJFR0lTVFJZX1BBU1NXT1JEPSQoZ3JlcCByZWdpc3RyeS1wYXNzd2Qg
L29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14i
XSpcKS4qL1wxL3AnKQogICAgaWYgW1sgLW4gIiRSRUdJU1RSWV9VU0VSTkFNRSIgJiYg
LW4gIiRSRUdJU1RSWV9QQVNTV09SRCIgXV07IHRoZW4KICAgICAgZG9ja2VyIGxvZ2lu
IC11ICRSRUdJU1RSWV9VU0VSTkFNRSAtcCAkUkVHSVNUUllfUEFTU1dPUkQgJFJFR0lT
VFJZX1VSSQogICAgZWxzZQogICAgICBlY2hvICJXYXJuaW5nOiB0aGUgcmVnaXN0cnkn
cyB1c2VybmFtZSBhbmQgcGFzc3dvcmQgYXJlIGludmFsaWQsIFNraXBwaW5nIERvY2tl
ciBsb2dpbi4iCiAgICBmaQogICAgICAgICAgICAjIElmIFJFR0lTVFJZX1VSSV9QQVRI
IC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQogICAgaWYgW1sgLWUg
LXAgODg4ODo4ODg4ICRSRUdJU1RSWV9VUklfUEFUSC9udmlkaWEvdGVuc29yZmxvdzoy
My4xMC10ZjItcHkzIC91c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9v
dCAtLWlwPSogLS1wb3J0PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAudG9r
```

Tabelle 3-3. TensorFlow-Container-Image (Fortsetzung)

Komponente	Beschreibung
	(Fortsetzung, siehe unten)


```
ZW49JycgLS1Ob3RlYm9va0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGly
PS93b3Jrc3BhY2UKCi0gcGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lv
bnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9l
eGl0KCkgewogICAgICBlY2hvICJFcnJvcjogJDEiID4mMgogICAgICB2bXRvb2xzZCAt
LWNtZCAiaW5mby1zZXQgZ3Vlc3RpbmZvLnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0
aW9uIGZhbHNlLCBETFdvcmtsb2FkRmFpbHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAg
fQoKICAgIGNoZWNrX3Byb3RvY29sKCkgewogICAgICBsb2NhbCBwcm94eV91cmw9JDEK
ICAgICAgc2hpZnQKICAgICAgbG9jYWwgc3VwcG9ydGVkX3Byb3RvY29scz0oIiRAIikK
ICAgICAgaWYgW1sgLW4gIiR7cHJveHlfdXJsfSIgXV07IHRoZW4KICAgICAgICBsb2Nh
bCBwcm90b2NvbD0kKGVjaG8gIiR7cHJveHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tp
ZiAoTkYgPiAxKSBwcmludCAkMTsgZWxzZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsg
LXogIiRwcm90b2NvbCIgXTsgdGhlbgogICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMg
cHJvdG9jb2wgcHJvdmlkZWQuIFNraXBwaW5nIHByb3RvY29sIGNoZWNrLiIKICAgICAg
ICAgIHJldHVybiAwCiAgICAgICAgZmkKICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNs
dWRlZD1mYWxzZQogICAgICAgIGZvciB2YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29s
c1tAXX0iOyBkbwogICAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9
IiBdXTsgdGhlbgogICAgICAgICAgICBwcm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAg
ICAgICAgIGJyZWFrCiAgICAgICAgICBmaQogICAgICAgIGRvbmUKICAgICAgICBpZiBb
WyAiJHtwcm90b2NvbF9pbmNsdWRlZH0iID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAg
ICBlcnJvcl9leGl0ICJVbnN1cHBvcnRlZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1
cHBvcnRlZCBwcm90b2NvbHMgYXJlOiAke3N1cHBvcnRlZF9wcm90b2NvbHNbKl19Igog
ICAgICAgIGZpCiAgICAgIGZpCiAgICB9CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0
ZWQgcHJvdG9jb2xzCiAgICBzZXRfcHJveHkoKSB7CiAgICAgIGxvY2FsIHN1cHBvcnRl
ZF9wcm90b2NvbHM9KCIkQCIpCgogICAgICBDT05GSUdfSlNPTl9CQVNFNjQ9JChncmVw
ICdjb25maWctanNvbicgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4q
b2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVj
aG8gJHtDT05GSUdfSlNPTl9CQVNFNjR9IHwgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAg
SFRUUF9QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0
dHBfcHJveHkgLy8gZW1wdHknKQogICAgICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIk
e0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0dHBzX3Byb3h5IC8vIGVtcHR5JykKICAg
ICAgaWYgW1sgJD8gLW5lIDAgfHwgKC16ICIke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXog
IiR7SFRUUFNfUFJPWFlfVVJMfSIpIF1dOyB0aGVuCiAgICAgICAgZWNobyAiSW5mbzog
VGhlIGNvbmZpZy1qc29uIHdhcyBwYXJzZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3
ZXJlIGZvdW5kLiIKICAgICAgICByZXR1cm4gMAogICAgICBmaQoKICAgICAgY2hlY2tf
cHJvdG9jb2wgIiR7SFRUUF9QUk9YWV9VUkx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xz
W0BdfSIKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7
c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iCgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBf
cHJveHknIC9ldGMvZW52aXJvbm1lbnQ7IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQg
aHR0cF9wcm94eT0ke0hUVFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBodHRwc19w
cm94eT0ke0hUVFBTX1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0k
e0hUVFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBT
X1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4w
LjAuMSIgPj4gL2V0Yy9lbnZpcm9ubWVudAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmly
b25tZW50CiAgICAgIGZpCiAgICAgIAogICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8g
dXNlIGEgcHJveHkKICAgICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2Nr
ZXIuc2VydmljZS5kCiAgICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVu
dD1cIkhUVFBfUFJPWFk9JHtIVFRQX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVu
dD1cIkhUVFBTX1BST1hZPSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25t
ZW50PVwiTk9fUFJPWFk9bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVt
ZC9zeXN0ZW0vZG9ja2VyLnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgIHN5c3RlbWN0
bCBkYWVtb24tcmVsb2FkCiAgICAgIHN5c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAg
ICAgZWNobyAiSW5mbzogZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5v
dyBjb25maWd1cmVkIHRvIHVzZSB0aGUgcHJveHkgc2V0dGluZ3MiCiAgICB9
```


was dem folgenden Skript im Klartextformat entspricht:

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
```

Tabelle 3-3. TensorFlow-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
  content: |
    #!/bin/bash
    set -eu
    source /opt/dlvm/utils.sh
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    set_proxy "http" "https" "socks5"

    DEFAULT_REG_URI="nvcr.io"
    REGISTRY_URI_PATH=$(grep registry-uri /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')

    if [[ -z "$REGISTRY_URI_PATH" ]]; then
      # If REGISTRY_URI_PATH is null or empty, use the default value
      REGISTRY_URI_PATH=$DEFAULT_REG_URI
      echo "REGISTRY_URI_PATH was empty. Using default:
$REGISTRY_URI_PATH"
    fi

    # If REGISTRY_URI_PATH contains '/', extract the URI part
    if [[ $REGISTRY_URI_PATH == *"/"* ]]; then
      REGISTRY_URI=$(echo "$REGISTRY_URI_PATH" | cut -d'/' -f1)
    else
      REGISTRY_URI=$REGISTRY_URI_PATH
    fi

    REGISTRY_USERNAME=$(grep registry-user /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    REGISTRY_PASSWORD=$(grep registry-passwd /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    if [[ -n "$REGISTRY_USERNAME" && -n "$REGISTRY_PASSWORD" ]];
then
      docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
$REGISTRY_URI
    else
      echo "Warning: the registry's username and password are
invalid, Skipping Docker login."
    fi

    docker run -d --gpus all -p 8888:8888 $REGISTRY_URI_PATH/
nvidia/tensorflow:23.10-tf2-py3 /usr/local/bin/jupyter lab --allow-
root --ip=* --port=8888 --no-browser --NotebookApp.token='' --
NotebookApp.allow_origin='*' --notebook-dir=/workspace

- path: /opt/dlvm/utils.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    error_exit() {
      echo "Error: $1" >&2
      vmtoolsd --cmd "info-set
guestinfo.vmservice.bootstrap.condition false, DLWorkloadFailure,
$1"
      exit 1
    }

    check_protocol() {
      local proxy_url=$1
      shift
      local supported_protocols=("$@")
      if [[ -n "${proxy_url}" ]]; then
        local protocol=$(echo "${proxy_url}" | awk -F '://' '{if
```

Tabelle 3-3. TensorFlow-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
(NF > 1) print $1; else print ""}')
        if [ -z "$protocol" ]; then
          echo "No specific protocol provided. Skipping protocol
check."
          return 0
        fi
        local protocol_included=false
        for var in "${supported_protocols[@]}"; do
          if [[ "${protocol}" == "${var}" ]]; then
            protocol_included=true
            break
          fi
        done
        if [[ "${protocol_included}" == false ]]; then
          error_exit "Unsupported protocol: ${protocol}. Supported
protocols are: ${supported_protocols[*]}"
        fi
      fi
    }

    # $@: list of supported protocols
    set_proxy() {
      local supported_protocols=("$@")

      CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
      CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} | base64 --decode)

      HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.http_proxy // empty')
      HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.https_proxy // empty')
      if [[ $? -ne 0 || (-z "${HTTP_PROXY_URL}" && -z "$
{HTTPS_PROXY_URL}") ]]; then
        echo "Info: The config-json was parsed, but no proxy
settings were found."
        return 0
      fi

      check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}"
      check_protocol "${HTTPS_PROXY_URL}" "$
{supported_protocols[@]}"

      if ! grep -q 'http_proxy' /etc/environment; then
        echo "export http_proxy=${HTTP_PROXY_URL}
        export https_proxy=${HTTPS_PROXY_URL}
        export HTTP_PROXY=${HTTP_PROXY_URL}
        export HTTPS_PROXY=${HTTPS_PROXY_URL}
        export no_proxy=localhost,127.0.0.1" >> /etc/environment
        source /etc/environment
      fi

      # Configure Docker to use a proxy
      mkdir -p /etc/systemd/system/docker.service.d
      echo "[Service]
      Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\"
      Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\"
      Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/
system/docker.service.d/proxy.conf
      systemctl daemon-reload
      systemctl restart docker
```

Tabelle 3-3. TensorFlow-Container-Image (Fortsetzung)

Komponente	Beschreibung
	<pre> echo "Info: docker and system environment are now configured to use the proxy settings" }</pre> ■ Einzeiliges Image. Codieren Sie es im base64-Format. <pre>docker run -d -p 8888:8888 nvcr.io/nvidia/tensorflow:*ngc_image_tag* /usr/local/bin/jupyter lab --allow-root --ip=* --port=8888 --no-browser --NotebookApp.token='' --NotebookApp.allow_origin='*' --notebook-dir=/workspace</pre> Geben Sie beispielsweise für „tensorflow:23.10-tf2-py3“ das folgende Skript im base64-Format an: <pre>ZG9ja2VyIHJ1biAtZCAtcCA4ODg4Ojg4ODggbnZjci5pby9udmlkaWEvdGVuc29yZmxvdzoyMy4xMC10ZjItcHkzIC91c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9vdCAtLWlwPSogLS1wb3J0PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAudG9rZW49JycgLS1Ob3RlYm9va0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGlyPS93b3Jrc3BhY2U=</pre> was dem folgenden Skript im Klartextformat entspricht: <pre>docker run -d -p 8888:8888 nvcr.io/nvidia/tensorflow:23.10-tf2-py3 /usr/local/bin/jupyter lab --allow-root --ip=* --port=8888 --no-browser --NotebookApp.token='' --NotebookApp.allow_origin='*' --notebook-dir=/workspace</pre> ■ Geben Sie die Installationseigenschaften des vGPU-Gasttreibers ein, wie z. B. `vgpu-license` und `nvidia-portal-api-key`. ■ Geben Sie nach Bedarf Werte für die Eigenschaften an, die für eine getrennte Umgebung erforderlich sind. Weitere Informationen finden Sie unter OVF-Eigenschaften von Deep Learning-VMs.
Ausgabe	■ Installationsprotokolle für den vGPU-Gasttreiber in `/var/log/vgpu-install.log`. Um zu überprüfen, ob der vGPU-Gasttreiber installiert ist, melden Sie sich über SSH bei der VM an und führen Sie den Befehl `nvidia-smi` aus. ■ Cloud-init-Skriptprotokolle in `/var/log/dl.log`. ■ TensorFlow-Container. Um zu überprüfen, ob der TensorFlow-Container ausgeführt wird, führen Sie die Befehle `sudo docker ps -a` und `sudo docker logs *container_id*` aus. ■ JupyterLab-Instanz, auf die Sie unter `http://*dl_vm_ip*:8888` zugreifen können. Stellen Sie im Terminal von JupyterLab sicher, dass die folgenden Funktionen im Notizbuch verfügbar sind: ■ Um zu überprüfen, ob JupyterLab auf die vGPU-Ressource zugreifen kann, führen Sie `nvidia-smi` aus. ■ Um sicherzustellen, dass die mit TensorFlow verbundenen Pakete installiert sind, führen Sie `pip show` aus.

DCGM Exporter

Sie können einer Deep Learning-VM mit einem DCGM Exporter (Data Center GPU Manager) verwenden, um den Zustand von GPUs zu überwachen und Metriken aus GPUs abzurufen, die von einer DL-Arbeitslast verwendet werden, indem Sie NVIDIA DCGM, Prometheus und Grafana verwenden.

Weitere Informationen finden Sie auf der Seite DCGM Exporter.

In einer Deep Learning-VM führen Sie den DCGM Exporter-Container zusammen mit einer DL-Arbeitslast aus, die KI-Vorgänge durchführt. Nachdem die Deep Learning-VM gestartet wurde, ist DCGM Exporter bereit, vGPU-Metriken zu erfassen und die Daten zur weiteren Überwachung und Visualisierung in eine andere Anwendung zu exportieren. Sie können die überwachte DL-Arbeitslast als Teil des cloud-init-Prozesses oder über die Befehlszeile ausführen, nachdem die virtuelle Maschine gestartet wurde.

Tabelle 3-4. DCGM Exporter-Container-Image

Komponente	Beschreibung
Container-Image	`nvcr.io/nvidia/k8s/dcgm-exporter:ngc_image_tag` Beispiel: `nvcr.io/nvidia/k8s/dcgm-exporter:3.2.5-3.1.8-ubuntu22.04` Informationen zu den DCGM Exporter-Container-Images, die für Deep Learning-VMs unterstützt werden, finden Sie unter Versionshinweise zu VMware Deep Learning VM.
Erforderliche Eingaben	Um eine DCGM Exporter-Arbeitslast bereitzustellen, müssen Sie die OVF-Eigenschaften für die Deep Learning-VM wie folgt festlegen: ■ Verwenden Sie eine der folgenden Eigenschaften, die spezifisch für das DCGM Exporter-Image sind. ■ Cloud-init-Skript. Codieren Sie es im base64-Format.

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    set -eu
    source /opt/dlvm/utils.sh
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    set_proxy "http" "https" "socks5"

    DEFAULT_REG_URI="nvcr.io"
    REGISTRY_URI_PATH=$(grep registry-uri /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')

    if [[ -z "$REGISTRY_URI_PATH" ]]; then
      # If REGISTRY_URI_PATH is null or empty, use the default value
      REGISTRY_URI_PATH=$DEFAULT_REG_URI
      echo "REGISTRY_URI_PATH was empty. Using default:
$REGISTRY_URI_PATH"
    fi

    # If REGISTRY_URI_PATH contains '/', extract the URI part
    if [[ $REGISTRY_URI_PATH == *"/"* ]]; then
      REGISTRY_URI=$(echo "$REGISTRY_URI_PATH" | cut -d'/' -f1)
    else
      REGISTRY_URI=$REGISTRY_URI_PATH
    fi

    REGISTRY_USERNAME=$(grep registry-user /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    REGISTRY_PASSWORD=$(grep registry-passwd /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    if [[ -n "$REGISTRY_USERNAME" && -n "$REGISTRY_PASSWORD" ]];
then
      docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
$REGISTRY_URI
    else
      echo "Warning: the registry's username and password are
invalid, Skipping Docker login."
    fi

    docker run -d --gpus all --cap-add SYS_ADMIN --rm -p 9400:9400
```

Tabelle 3-4. DCGM Exporter-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
$REGISTRY_URI_PATH/nvidia/k8s/dcgm-exporter:ngc_image_tag

- path: /opt/dlvm/utils.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    error_exit() {
      echo "Error: $1" >&2
      vmtoolsd --cmd "info-set
guestinfo.vmservice.bootstrap.condition false, DLWorkloadFailure,
$1"
      exit 1
    }

    check_protocol() {
      local proxy_url=$1
      shift
      local supported_protocols=("$@")
      if [[ -n "${proxy_url}" ]]; then
        local protocol=$(echo "${proxy_url}" | awk -F '://' '{if
(NF > 1) print $1; else print ""}')
        if [ -z "$protocol" ]; then
          echo "No specific protocol provided. Skipping protocol
check."
          return 0
        fi
        local protocol_included=false
        for var in "${supported_protocols[@]}"; do
          if [[ "${protocol}" == "${var}" ]]; then
            protocol_included=true
            break
          fi
        done
        if [[ "${protocol_included}" == false ]]; then
          error_exit "Unsupported protocol: ${protocol}. Supported
protocols are: ${supported_protocols[*]}"
        fi
      fi
    }

    # $@: list of supported protocols
    set_proxy() {
      local supported_protocols=("$@")

      CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
      CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} | base64 --decode)

      HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.http_proxy // empty')
      HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.https_proxy // empty')
      if [[ $? -ne 0 || (-z "${HTTP_PROXY_URL}" && -z "$
{HTTPS_PROXY_URL}") ]]; then
        echo "Info: The config-json was parsed, but no proxy
settings were found."
        return 0
      fi

      check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}"
      check_protocol "${HTTPS_PROXY_URL}" "$
```

Tabelle 3-4. DCGM Exporter-Container-Image (Fortsetzung)

Komponente	Beschreibung
	(Fortsetzung, siehe unten)

```
{supported_protocols[@]}"

      if ! grep -q 'http_proxy' /etc/environment; then
        echo "export http_proxy=${HTTP_PROXY_URL}
        export https_proxy=${HTTPS_PROXY_URL}
        export HTTP_PROXY=${HTTP_PROXY_URL}
        export HTTPS_PROXY=${HTTPS_PROXY_URL}
        export no_proxy=localhost,127.0.0.1" >> /etc/environment
        source /etc/environment
      fi

      # Configure Docker to use a proxy
      mkdir -p /etc/systemd/system/docker.service.d
      echo "[Service]
      Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\"
      Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\"
      Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/
system/docker.service.d/proxy.conf
      systemctl daemon-reload
      systemctl restart docker

      echo "Info: docker and system environment are now configured
to use the proxy settings"
    }
```

Geben Sie beispielsweise für eine Deep Learning-VM mit dem vorinstallierten DCGM Exporter „dcgm-exporter-Instanz:3.2.5-3.1.8-ubuntu22.04“ das folgende Skript im base64-Format an


```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCiAg
ICAKICAgIERFRkFVTFRfUkVHX1VSST0ibnZjci5pbyIKICAgIFJFR0lTVFJZX1VSSV9Q
QVRIPSQoZ3JlcCByZWdpc3RyeS11cmkgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2Vk
IC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIGlmIFtbIC16
ICIkUkVHSVNUUllfVVJJX1BBVEgiIF1dOyB0aGVuCiAgICAgICMgSWYgUkVHSVNUUllf
VVJJX1BBVEggaXMgbnVsbCBvciBlbXB0eSwgdXNlIHRoZSBkZWZhdWx0IHZhbHVlCiAg
ICAgIFJFR0lTVFJZX1VSSV9QQVRIPSRERUZBVUxUX1JFR19VUkkKICAgICAgZWNobyAi
UkVHSVNUUllfVVJJX1BBVEggd2FzIGVtcHR5LiBVc2luZyBkZWZhdWx0OiAkUkVHSVNU
UllfVVJJX1BBVEgiCiAgICBmaQogICAgCiAgICAjIElmIFJFR0lTVFJZX1VSSV9QQVRI
IGNvbnRhaW5zICcvJywgZXh0cmFjdCB0aGUgVVJJIHBhcnQKICAgIGlmIFtbICRSRUdJ
U1RSWV9VUklfUEFUSCA9PSAqIi8iKiBdXTsgdGhlbgogICAgICBSRUdJU1RSWV9VUkk9
JChlY2hvICIkUkVHSVNUUllfVVJJX1BBVEgiIHwgY3V0IC1kJy8nIC1mMSkKICAgIGVs
c2UKICAgICAgUkVHSVNUUllfVVJJPSRSRUdJU1RSWV9VUklfUEFUSAogICAgZmkKICAK
ICAgIFJFR0lTVFJZX1VTRVJOQU1FPSQoZ3JlcCByZWdpc3RyeS11c2VyIC9vcHQvZGx2
bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9c
MS9wJykKICAgIFJFR0lTVFJZX1BBU1NXT1JEPSQoZ3JlcCByZWdpc3RyeS1wYXNzd2Qg
L29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14i
XSpcKS4qL1wxL3AnKQogICAgaWYgW1sgLW4gIiRSRUdJU1RSWV9VU0VSTkFNRSIgJiYg
LW4gIiRSRUdJU1RSWV9QQVNTV09SRCIgXV07IHRoZW4KICAgICAgZG9ja2VyIGxvZ2lu
IC11ICRSRUdJU1RSWV9VU0VSTkFNRSAtcCAkUkVHSVNUUllfUEFTU1dPUkQgJFJFR0lT
VFJZX1VSSQogICAgZWxzZQogICAgICBlY2hvICJXYXJuaW5nOiB0aGUgcmVnaXN0cnkn
cyB1c2VybmFtZSBhbmQgcGFzc3dvcmQgYXJlIGludmFsaWQsIFNraXBwaW5nIERvY2tl
ciBsb2dpbi4iCiAgICBmaQoKICAgIGRvY2tlciBydW4gLWQgLS1ncHVzIGFsbCAtLWNh
cC1hZGQgU1lTX0FETUlOIC0tcm0gLXAgOTQwMDo5NDAwICRSRUdJU1RSWV9VUklfUEFU
SC9udmlkaWEvazhzL2RjZ20tZXhwb3J0ZXI6My4yLjUtMy4xLjgtdWJ1bnR1MjIuMDQK
Ci0gcGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1Jwog
IGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAg
```

Tabelle 3-4. DCGM Exporter-Container-Image (Fortsetzung)

Komponente	Beschreibung
	ICBlY2hvICJFcnJvcjogJDEiID4mMgogICAgICB2bXRvb2xzZCAtLWNtZCAiaW5mby1z ZXQgZ3Vlc3RpbmZvLnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIGZhbHNlLCBE TFdvcmtsb2FkRmFpbHVyZSwgJDEiCiAgICAgICAgV4aXQgMQogICAgfQoKICAgIGNoZWNr X3Byb3RvY29sKCkgewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQK ICAgICAgbG9jYWwgc3VwcG9ydGVkX3Byb3RvY29scz0oIiRAIikKICAgICAgaWYgW1sg LW4gIiR7cHJveHlfdXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm90b2NvbD0k KGVjaG8gIiR7cHJveHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBw cmludCAkMTsgZWxzZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2Nv bCIgXTsgdGhlbgogICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMgcHJvdG9jb2wgcHJv dmlkZWQuIFNraXBwaW5nIHByb3RvY29sIGNoZWNrLiKICAgICAgICAgIHJldHVybiAw CiAgICAgICAgZmkKICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQog ICAgICAgIGZvciB2YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwog ICAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9IiBdXTsgdGhlbgog ICAgICAgICAgICBwcm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgIGJyZWFr CiAgICAgICAgICBmaQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2Nv bF9pbmNsdWRlZH0iID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0 ICJVbnN1cHBvcnRlZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1cHBvcnRlZCBwcm90 b2NvbHMgYXJlOiAke3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgIGZpCiAg ICAgIGZpCiAgICB9CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xz CiAgICBzZXRfcHJveHkoKSB7CiAgICAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9 KCIkQCIpCgogICAgICBDT05GSUdfSlNPTl9CQVNFNjQ9JChncmVwICdjb25maWctanNv bicgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9Ilwo W14iXSpcKS4qL1wxL3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdf SlNPTl9CQVNFNjR9IHwgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgSFRUUF9QUk9YWV9V Ukw9JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0dHBfcHJveHkgLy8g ZW1wdHknKQogICAgICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09O fSIgfCBqcSAtciAnLmh0dHBzX3Byb3h5IC8vIGVtcHR5JykKICAgICAgaWYgW1sgJD8g LW5lIDAgfHwgKC16ICIke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJP WFlfVVJMfSIpIF1dOyB0aGVuCiAgICAgICAgZWNobyAiSW5mbzogVGhlIGNvbmZpZy1q c29uIHdhcyBwYXJzZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kLiIK ICAgICAgICByZXR1cm4gMAogICAgICBmaQoKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7 SFRUUF9QUk9YWV9VUkx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAg Y2hlY2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVkX3By b3RvY29sc1tAXX0iCgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldGMv ZW52aXJvbm1lbnQ7IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0k e0hUVFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBT X1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJPWFlf VVJMfQogICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH0K ICAgICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0 Yy9lbnZpcm9ubWVudAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAgICAg IGZpCiAgICAgICAgIAogICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJveHk ICAgICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5k CiAgICAgICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJP WFk9JHtIVFRQX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BS T1hZPSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgICAgIEVudmlyb25tZW50PVwiTk9fUFJP WFk9bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9j a2VyLnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVs b2FkCiAgICAgICAgIHN5c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgICAgZWNobyAiSW5m bzogZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1cmVk IHRvIHVzZSB0aGUgcHJveHkgc2V0dGluZ3MiCiAgICB9

was dem folgenden Skript im Klartextformat entspricht:

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    set -eu
```

Tabelle 3-4. DCGM Exporter-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
    source /opt/dlvm/utils.sh
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    set_proxy "http" "https" "socks5"

    DEFAULT_REG_URI="nvcr.io"
    REGISTRY_URI_PATH=$(grep registry-uri /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')

    if [[ -z "$REGISTRY_URI_PATH" ]]; then
      # If REGISTRY_URI_PATH is null or empty, use the default value
      REGISTRY_URI_PATH=$DEFAULT_REG_URI
      echo "REGISTRY_URI_PATH was empty. Using default:
$REGISTRY_URI_PATH"
    fi

    # If REGISTRY_URI_PATH contains '/', extract the URI part
    if [[ $REGISTRY_URI_PATH == *"/"* ]]; then
      REGISTRY_URI=$(echo "$REGISTRY_URI_PATH" | cut -d'/' -f1)
    else
      REGISTRY_URI=$REGISTRY_URI_PATH
    fi

    REGISTRY_USERNAME=$(grep registry-user /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    REGISTRY_PASSWORD=$(grep registry-passwd /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    if [[ -n "$REGISTRY_USERNAME" && -n "$REGISTRY_PASSWORD" ]];
then
      docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
$REGISTRY_URI
    else
      echo "Warning: the registry's username and password are
invalid, Skipping Docker login."
    fi

    docker run -d --gpus all --cap-add SYS_ADMIN --rm -p 9400:9400
$REGISTRY_URI_PATH/nvidia/k8s/dcgm-exporter:3.2.5-3.1.8-ubuntu22.04

- path: /opt/dlvm/utils.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    error_exit() {
      echo "Error: $1" >&2
      vmtoolsd --cmd "info-set
guestinfo.vmservice.bootstrap.condition false, DLWorkloadFailure,
$1"
      exit 1
    }

    check_protocol() {
      local proxy_url=$1
      shift
      local supported_protocols=("$@")
      if [[ -n "${proxy_url}" ]]; then
        local protocol=$(echo "${proxy_url}" | awk -F '://' '{if
(NF > 1) print $1; else print ""}')
        if [ -z "$protocol" ]; then
          echo "No specific protocol provided. Skipping protocol
check."
```

Tabelle 3-4. DCGM Exporter-Container-Image (Fortsetzung)

Komponente	Beschreibung
	<pre> return 0 fi local protocol_included=false for var in "${supported_protocols[@]}"; do if [["${protocol}" == "${var}"]]; then protocol_included=true break fi done if [["${protocol_included}" == false]]; then error_exit "Unsupported protocol: ${protocol}. Supported protocols are: ${supported_protocols[*]}" fi fi } # $@: list of supported protocols set_proxy() { local supported_protocols=("$@") CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml | sed -n 's/.*oe:value="\([^"]*\).*/\1/p') CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} | base64 --decode) HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r '.http_proxy // empty') HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r '.https_proxy // empty') if [[$? -ne 0 || (-z "${HTTP_PROXY_URL}" && -z "$ {HTTPS_PROXY_URL}")]]; then echo "Info: The config-json was parsed, but no proxy settings were found." return 0 fi check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}" check_protocol "${HTTPS_PROXY_URL}" "$ {supported_protocols[@]}" if ! grep -q 'http_proxy' /etc/environment; then echo "export http_proxy=${HTTP_PROXY_URL} export https_proxy=${HTTPS_PROXY_URL} export HTTP_PROXY=${HTTP_PROXY_URL} export HTTPS_PROXY=${HTTPS_PROXY_URL} export no_proxy=localhost,127.0.0.1" >> /etc/environment source /etc/environment fi # Configure Docker to use a proxy mkdir -p /etc/systemd/system/docker.service.d echo "[Service] Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\" Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\" Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/ system/docker.service.d/proxy.conf systemctl daemon-reload systemctl restart docker</pre>

Tabelle 3-4. DCGM Exporter-Container-Image (Fortsetzung)

Komponente	Beschreibung
	echo "Info: docker and system environment are now configured to use the proxy settings" } **Hinweis** Sie können auch die Anweisungen zum Ausführen der DL-Arbeitslast, deren GPU-Leistung Sie mit DCGM Exporter messen möchten, zum cloud-init-Skript hinzufügen. ■ Einzeiliges Image. Codieren Sie es im base64-Format. `docker run -d --gpus all --cap-add SYS_ADMIN --rm -p 9400:9400 nvcr.io/nvidia/k8s/dcgm-exporter:`*`ngc_image_tag`*`-ubuntu22.04` Geben Sie beispielsweise für „dcgm-exporter:3.2.5-3.1.8-ubuntu22.04“ das folgende Skript im base64-Format an: `ZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tY2FwLWFkZCBTWVNfQURNSU4gLS1ybSAtcCA5NDAwOjk0MDAgbnZjci5pby9udmlkaWEvazhzL2RjZ20tZXhwb3J0ZXI6My4yLjUtMy4xLjgtdWJ1bnR1MjIuMDQ=` was dem folgenden Skript im Klartextformat entspricht: `docker run -d --gpus all --cap-add SYS_ADMIN --rm -p 9400:9400 nvcr.io/nvidia/k8s/dcgm-exporter:3.2.5-3.1.8-ubuntu22.04` ■ Geben Sie die Installationseigenschaften des vGPU-Gasttreibers ein, wie z. B. `vgpu-license` und `nvidia-portal-api-key`. ■ Geben Sie nach Bedarf Werte für die Eigenschaften an, die für eine getrennte Umgebung erforderlich sind. Weitere Informationen finden Sie unter OVF-Eigenschaften von Deep Learning-VMs.
Ausgabe	■ Installationsprotokolle für den vGPU-Gasttreiber in `/var/log/vgpu-install.log`. Um zu überprüfen, ob der vGPU-Gasttreiber installiert ist, melden Sie sich über SSH bei der VM an und führen Sie den Befehl `nvidia-smi` aus. ■ Cloud-init-Skriptprotokolle in `/var/log/dl.log`. ■ DCGM Exporter, auf den Sie unter `http://`*`dl_vm_ip`*`:9400` zugreifen können. Anschließend führen Sie in der Deep Learning-VM eine DL-Arbeitslast aus und visualisieren die Daten auf einer anderen virtuellen Maschine mithilfe von Prometheus bei `http://`*`visualization_vm_ip`*`:9090` und Grafana bei `http://`*`visualization_vm_ip`*`:3000`.

Ausführen einer DL-Arbeitslast auf der Deep-Lean-VM

Führen Sie die DL-Arbeitslast aus, für die Sie vGPU-Metriken erfassen möchten, und exportieren Sie die Daten zur weiteren Überwachung und Visualisierung in eine andere Anwendung.

1 Melden Sie sich bei der Deep Learning-VM als **vmware** über SSH an.

2 Fügen Sie das Benutzerkonto **vmware** zur Gruppe **docker** hinzu, indem Sie den folgenden Befehl ausführen.

```
sudo usermod -aG docker ${USER}
```

3 Führen Sie den Container für die DL-Arbeitslast aus und ziehen Sie ihn aus dem NVIDIA NGC-Katalog oder aus einer lokalen Containerregistrierung.

   So führen Sie beispielsweise den folgenden Befehl aus, um das Tensorflow-Image 23.10-tf2-py3 von NVIDIA NGC auszuführen:

```
docker run -d -p 8888:8888 nvcr.io/nvidia/tensorflow:23.10-tf2-py3 /usr/local/bin/
jupyter lab --allow-root --ip=* --port=8888 --no-browser --NotebookApp.token='' --
NotebookApp.allow_origin='*' --notebook-dir=/workspace
```

4 Beginnen Sie mit der Verwendung der DL-Arbeitslast für die KI-Entwicklung.

Installieren von Prometheus und Grafana

Sie können die vGPU-Metriken von der DCGM Exporter-VM auf einer virtuellen Maschine, auf der Prometheus und Grafana ausgeführt wird, visualisieren und überwachen.

1 Erstellen Sie eine Visualisierungs-VM mit installierter Docker Community Engine.

2 Stellen Sie über SSH eine Verbindung zur VM her und erstellen Sie eine YAML-Datei für Prometheus.

```
$ cat > prometheus.yml << EOF
global:
  scrape_interval: 15s
  external_labels:
    monitor: 'codelab-monitor'
scrape_configs:
  - job_name: 'dcgm'
    scrape_interval: 5s
    metrics_path: /metrics
    static_configs:
      - targets: [dl_vm_with_dcgm_exporter_ip:9400']
EOF
```

3 Erstellen Sie einen Datenpfad.

```
$ mkdir grafana_data prometheus_data && chmod 777 grafana_data prometheus_data
```

4 Erstellen Sie eine Docker-Erstellungsdatei, um Prometheus und Grafana zu installieren.

```
$ cat > compose.yaml << EOF
services:
  prometheus:
    image: prom/prometheus:v2.47.2
    container_name: "prometheus0"
    restart: always
    ports:
      - "9090:9090"
    volumes:
      - "./prometheus.yml:/etc/prometheus/prometheus.yml"
      - "./prometheus_data:/prometheus"
  grafana:
    image: grafana/grafana:10.2.0-ubuntu
```

```
    container_name: "grafana0"
    ports:
      - "3000:3000"
    restart: always
    volumes:
      - "./grafana_data:/var/lib/grafana"
EOF
```

5 Starten Sie die Prometheus- und Grafana-Container.

```
$ sudo docker compose up -d
```

Anzeigen von vGPU-Metriken in Prometheus

Sie können auf Prometheus unter `http://visualization-vm-ip:9090` zugreifen. Sie können die folgenden vGPU-Informationen in der Prometheus-Benutzeroberfläche anzeigen:

Informationen	Abschnitt der Benutzeroberfläche
vGPU-Rohmetriken aus der Deep Learning-VM	**Status > Ziel** Um die vGPU-Rohmetriken aus der Deep Learning-VM anzuzeigen, klicken Sie auf den Endpoint-Eintrag.
Diagrammausdrücke	1 Klicken Sie in der Hauptnavigationsleiste auf die Registerkarte **Diagramm**. 2 Geben Sie einen Ausdruck ein und klicken Sie auf **Ausführen**

Weitere Informationen zur Verwendung von Prometheus finden Sie in der Prometheus-Dokumentation.

Visualisieren von Metriken in Grafana

Legen Sie Prometheus als Datenquelle für Grafana fest und visualisieren Sie die vGPU-Metriken aus der Deep Learning-VM in einem Dashboard.

1 Greifen Sie unter `http://visualization-vm-ip:3000` auf Grafana zu, indem Sie den Standardbenutzernamen **admin** und das Kennwort `admin` verwenden.

2 Fügen Sie Prometheus als erste Datenquelle hinzu und verbinden Sie sich mit *visualization-vm-ip* auf Port 9090.

3 Erstellen Sie ein Dashboard mit den vGPU-Metriken.

Weitere Informationen zum Konfigurieren eines Dashboards mithilfe einer Prometheus-Datenquelle finden Sie in der Grafana-Dokumentation.

Triton Inference Server

Sie können eine Deep Learning-VM mit einem Triton Inference Server verwenden, um ein Modell-Repository zu laden und Rückschlussanforderungen zu erhalten.

Weitere Informationen finden Sie auf der Seite Triton Inference Server.

Tabelle 3-5. Triton Inference Server-Container-Image

Komponente	Beschreibung
Container-Image	`nvcr.io/nvidia/tritonserver:ngc_image_tag` Beispiel: `nvcr.io/nvidia/tritonserver:23.10-py3` Informationen zu den Triton Inference Server-Container-Images, die für Deep Learning-VMs unterstützt werden, finden Sie unter Versionshinweise zu VMware Deep Learning VM.
Erforderliche Eingaben	Um eine Triton Inference Server-Arbeitslast bereitzustellen, müssen Sie die OVF-Eigenschaften für die Deep Learning-VM wie folgt festlegen: ■ Verwenden Sie eine der folgenden Eigenschaften, die für das Triton Inference Server-Image spezifisch sind. ■ Cloud-init-Skript. Codieren Sie es im base64-Format.

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    set -eu
    source /opt/dlvm/utils.sh
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    set_proxy "http" "https" "socks5"

    DEFAULT_REG_URI="nvcr.io"
    REGISTRY_URI_PATH=$(grep registry-uri /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')

    if [[ -z "$REGISTRY_URI_PATH" ]]; then
      # If REGISTRY_URI_PATH is null or empty, use the default value
      REGISTRY_URI_PATH=$DEFAULT_REG_URI
      echo "REGISTRY_URI_PATH was empty. Using default:
$REGISTRY_URI_PATH"
    fi

    # If REGISTRY_URI_PATH contains '/', extract the URI part
    if [[ $REGISTRY_URI_PATH == *"/"* ]]; then
      REGISTRY_URI=$(echo "$REGISTRY_URI_PATH" | cut -d'/' -f1)
    else
      REGISTRY_URI=$REGISTRY_URI_PATH
    fi

    REGISTRY_USERNAME=$(grep registry-user /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    REGISTRY_PASSWORD=$(grep registry-passwd /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    if [[ -n "$REGISTRY_USERNAME" && -n "$REGISTRY_PASSWORD" ]];
then
      docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
$REGISTRY_URI
    else
      echo "Warning: the registry's username and password are
invalid, Skipping Docker login."
    fi

    docker run -d --gpus all --rm -p 8000:8000 -p
```

Tabelle 3-5. Triton Inference Server-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
8001:8001 -p 8002:8002 -v /home/vmware/model_repository:/models
$REGISTRY_URI_PATH/nvidia/tritonserver:ngc_image_tag tritonserver --
model-repository=/models --model-control-mode=poll

- path: /opt/dlvm/utils.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    error_exit() {
      echo "Error: $1" >&2
      vmtoolsd --cmd "info-set
guestinfo.vmservice.bootstrap.condition false, DLWorkloadFailure,
$1"
      exit 1
    }

    check_protocol() {
      local proxy_url=$1
      shift
      local supported_protocols=("$@")
      if [[ -n "${proxy_url}" ]]; then
        local protocol=$(echo "${proxy_url}" | awk -F '://' '{if
(NF > 1) print $1; else print ""}')
        if [ -z "$protocol" ]; then
          echo "No specific protocol provided. Skipping protocol
check."
          return 0
        fi
        local protocol_included=false
        for var in "${supported_protocols[@]}"; do
          if [[ "${protocol}" == "${var}" ]]; then
            protocol_included=true
            break
          fi
        done
        if [[ "${protocol_included}" == false ]]; then
          error_exit "Unsupported protocol: ${protocol}. Supported
protocols are: ${supported_protocols[*]}"
        fi
      fi
    }

    # $@: list of supported protocols
    set_proxy() {
      local supported_protocols=("$@")

      CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
      CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} | base64 --decode)

      HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.http_proxy // empty')
      HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.https_proxy // empty')
      if [[ $? -ne 0 || (-z "${HTTP_PROXY_URL}" && -z "$
{HTTPS_PROXY_URL}") ]]; then
        echo "Info: The config-json was parsed, but no proxy
settings were found."
        return 0
      fi
```

Tabelle 3-5. Triton Inference Server-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
      check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}"
      check_protocol "${HTTPS_PROXY_URL}" "$
{supported_protocols[@]}"

      if ! grep -q 'http_proxy' /etc/environment; then
        echo "export http_proxy=${HTTP_PROXY_URL}
        export https_proxy=${HTTPS_PROXY_URL}
        export HTTP_PROXY=${HTTP_PROXY_URL}
        export HTTPS_PROXY=${HTTPS_PROXY_URL}
        export no_proxy=localhost,127.0.0.1" >> /etc/environment
        source /etc/environment
      fi

      # Configure Docker to use a proxy
      mkdir -p /etc/systemd/system/docker.service.d
      echo "[Service]
      Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\"
      Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\"
      Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/
system/docker.service.d/proxy.conf
      systemctl daemon-reload
      systemctl restart docker

      echo "Info: docker and system environment are now configured
to use the proxy settings"
    }
```

Geben Sie beispielsweise für „tritonserver:23.10-py3“ das folgende Skript im base64-Format an:

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkb3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCgog
ICAgREVGQVVMVF9SRUdfVVJJPSJudmNyLmlvIgogICAgUkVHSVNUUllfVVJJX1BBVEg9
JChncmVwIHJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4g
J3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCgogICAgaWYgW1sgLXogIiRS
RUdJU1RSWV9VUklfUEFUSCIgXV07IHRoZW4KICAgICAgIyBJZiBSRUdJU1RSWV9VUklf
UEFUSCBpcyBudWxsIG9yIGVtcHR5LCB1c2UgdGhlIGRlZmF1bHQgdmFsdWUKICAgICAg
UkVHSVNUUllfVVJJX1BBVEg9JERFRkFVTFRfUkVHX1VSSQogICAgICBlY2hvICJSRUdJ
U1RSWV9VUklfUEFUSCB3YXMgZW1wdHkuIFVzaW5nIGRlZmF1bHQ6ICRSRUdJU1RSWV9V
UklfUEFUSCIKICAgIGZpCiAgICAKICAgICMgSWYgUkVHSVNUUllfVVJJX1BBVEggY29u
dGFpbnMgJy8nLCBleHRyYWN0IHRoZSBVUkkgcGFydAogICAgaWYgW1sgJFJFR0lTVFJZ
X1VSSV9QQVRIID09ICoiLyIqIF1dOyB0aGVuCiAgICAgIFJFR0lTVFJZX1VSST0kKGVj
aG8gIiRSRUdJU1RSWV9VUklfUEFUSCIgfCBjdXQgLWQnLycgLWYxKQogICAgZWxzZQog
ICAgICBSRUdJU1RSWV9VUkk9JFJFR0lTVFJZX1VSSV9QQVRICiAgICBmaQogICAgUkVH
SVNUUllfVVNFUk5BTUU9JChncmVwIHJlZ2lzdHJ5LXVzZXIgL29wdC9kbHZtL292
Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3An
KQogICAgUkVHSVNUUllfUEFTU1dPUkQ9JChncmVwIHJlZ2lzdHJ5LXBhc3N3ZCAvb3B0
L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwp
LiovXDEvcCcpCiAgICBpZiBbWyAtbiAiJFJFR0lTVFJZX1VTRVJOQU1FIiAmJiAtbiAi
JFJFR0lTVFJZX1BBU1NXT1JEIiBdXTsgdGhlbgogICAgICBkb2NrZXIgbG9naW4gLXUg
JFJFR0lTVFJZX1VTRVJOQU1FIC1wICRSRUdJU1RSWV9QQVNTV09SRCAkUkVHSVNUUllf
VVJJCiAgICBlbHNlCiAgICAgIGVjaG8gIldhcm5pbmc6IHRoZSByZWdpc3RyeSdzIHVz
ZXJuYW1lIGFuZCBwYXNzd29yZCBhcmUgaW52YWxpZCwgU2tpcHBpbmcgRG9ja2VyIGxv
Z2luLiIKICAgIGZpCgogICAgZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tcm0gLXAg
ODAwMDo4MDAwIC1wIDgwMDE6ODAwMSAtcA4MDAyOjgwMDIgLXYgL2hvbWUvdm13YXJl
L21vZGVsX3JlcG9zaXRvcnk6L21vZGVscyAkUkVHSVNUUllfVVJJX1BBVEgvbnZpZGlh
L3RyaXRvbnNlcnZlcjoyMy4xMC1weTMgdHJpdG9uc2VydmVyIC0tbW9kZWwtcmVwb3Np
```

Tabelle 3-5. Triton Inference Server-Container-Image (Fortsetzung)

Komponente	Beschreibung
	dG9yeT0vbW9kZWxzIC0tbW9kZWwtY29udHJvbC1tb2RlPXBvbGwKCi0gcGF0aDogL29w dC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwK ICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAgICBlY2hvICJFcnJv cjogJDEiID4mMgogICAgICB2bXRvb2xzZCAtLWNtZCAiaW5mby1zZXQgZ3Vlc3RpbmZv LnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIGZhbHNlLCBETFdvcmtsb2FkRmFp bHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgfQoKICAgIGNoZWNrX3Byb3RvY29sKCkg ewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQKICAgICAgbG9jYWwg c3VwcG9ydGVkX3Byb3RvY29scz0oIiRAIikKICAgICAgaWYgW1sgLW4gIiR7cHJveHlf dXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm90b2NvbD0kKGVjaG8gIiR7cHJv eHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBwcmludCAkMTsgZWxz ZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2NvbCIgXTsgdGhlbgog ICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMgcHJvdG9jb2wgcHJvdmlkZWQuIFNraXBw aW5nIHByb3RvY29sIGNoZWNrLiKICAgICAgICAgIHJldHVybiAwCiAgICAgICAgZmkK ICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQogICAgICAgIGZvciB2 YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwogICAgICAgICAgaWYg W1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9IiBdXTsgdGhlbgogICAgICAgICAgICBw cm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgIGJyZWFrCiAgICAgICAgICBm aQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2NvbF9pbmNsdWRlZH0i ID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0ICJVbnN1cHBvcnRl ZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1cHBvcnRlZCBwcm90b2NvbHMgYXJlOiAk e3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgICAgZXhpdCAxCiAgICAgICAg CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xzCiAgICBBzZXRfcHJv eHkoKSB7CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIpCgogICAg ICBDT05GSUdfSlNPTl9CQVNFNjQ9JChncmVwICdjb25maWctanNvbicgL29wdC9kbHZt L292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wx L3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdfSlNPTl9CQVNFNjR9 IHwgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgSFRUUF9QUk9YWV9VUkw9JChlY2hvICIk e0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0dHBfcHJveHkgLy8gZW1wdHknKQogICAg ICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqcSAtciAn Lmh0dHBzX3Byb3h5IC8vIGVtcHR5JykKICAgICAgaWYgW1sgJD8gLW5lIDAgfHwgKC16 ICIke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJPWFlfVVJMfSIpIF1d OyB0aGVuCiAgICAgICAgZWNobyAiSW5mbzogVGhlIGNvbmZpZy1qc29uIHdhcyBwYXJz ZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kLiIKICAgICAgICAgIyAx cm4gMAogICAgICBmaQoKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUF9QUk9YWV9V Ukx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAgY2hlY2tfcHJvdG9j b2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0i CgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldGMvZW52aXJvbm1lbnQ7 IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0ke0hUVFBfUFJPWFlf VVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBTX1BST1hZX1VSTH0K ICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJPWFlfVVJMfQogICAgICAg IGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH0KICAgICAgICBleHBv cnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0Yy9lbnZpcm9ubWVu dAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAgICAgICAgZpCiAgICAgICAg ICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJveHkKICAgICAgICAgbWtaXIg LXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5kCiAgICAgICAgIGVjaG8g IltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJPWFk9JHtIVFRQX1BS T1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BST1hZPSR7SFRUUFNf UFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25tZW50PVwiTk9fUFJPWFk9bG9jYWxob3N0 LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9ja2VyLnNlcnZpY2Uu ZC9wcm94eS5jb25mCiAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVsb2FkCiAgICAgIHN5 c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAiSW5mbzogZG9ja2VyIGFu ZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1cmVkIHRvIHVzZSB0aGUg cHJveHkgc2V0dGluZ3MiCiAgICB9

was dem folgenden Skript im Klartextformat entspricht:

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
```

Tabelle 3-5. Triton Inference Server-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
  content: |
    #!/bin/bash
    set -eu
    source /opt/dlvm/utils.sh
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    set_proxy "http" "https" "socks5"

    DEFAULT_REG_URI="nvcr.io"
    REGISTRY_URI_PATH=$(grep registry-uri /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')

    if [[ -z "$REGISTRY_URI_PATH" ]]; then
      # If REGISTRY_URI_PATH is null or empty, use the default value
      REGISTRY_URI_PATH=$DEFAULT_REG_URI
      echo "REGISTRY_URI_PATH was empty. Using default:
$REGISTRY_URI_PATH"
    fi

    # If REGISTRY_URI_PATH contains '/', extract the URI part
    if [[ $REGISTRY_URI_PATH == *"/"* ]]; then
      REGISTRY_URI=$(echo "$REGISTRY_URI_PATH" | cut -d'/' -f1)
    else
      REGISTRY_URI=$REGISTRY_URI_PATH
    fi

    REGISTRY_USERNAME=$(grep registry-user /opt/dlvm/ovf-env.xml |
sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    REGISTRY_PASSWORD=$(grep registry-passwd /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
    if [[ -n "$REGISTRY_USERNAME" && -n "$REGISTRY_PASSWORD" ]];
then
      docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
$REGISTRY_URI
    else
      echo "Warning: the registry's username and password are
invalid, Skipping Docker login."
    fi

    docker run -d --gpus all --rm -p 8000:8000 -p
8001:8001 -p 8002:8002 -v /home/vmware/model_repository:/models
$REGISTRY_URI_PATH/nvidia/tritonserver:23.10-py3 tritonserver --
model-repository=/models --model-control-mode=poll

- path: /opt/dlvm/utils.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    error_exit() {
      echo "Error: $1" >&2
      vmtoolsd --cmd "info-set
guestinfo.vmservice.bootstrap.condition false, DLWorkloadFailure,
$1"
      exit 1
    }

    check_protocol() {
      local proxy_url=$1
      shift
      local supported_protocols=("$@")
      if [[ -n "${proxy_url}" ]]; then
        local protocol=$(echo "${proxy_url}" | awk -F '://' '{if
```

Tabelle 3-5. Triton Inference Server-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
(NF > 1) print $1; else print ""}')
        if [ -z "$protocol" ]; then
          echo "No specific protocol provided. Skipping protocol
check."
          return 0
        fi
        local protocol_included=false
        for var in "${supported_protocols[@]}"; do
          if [[ "${protocol}" == "${var}" ]]; then
            protocol_included=true
            break
          fi
        done
        if [[ "${protocol_included}" == false ]]; then
          error_exit "Unsupported protocol: ${protocol}. Supported
protocols are: ${supported_protocols[*]}"
        fi
      fi
    }

    # $@: list of supported protocols
    set_proxy() {
      local supported_protocols=("$@")

      CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
      CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} | base64 --decode)

      HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.http_proxy // empty')
      HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r
'.https_proxy // empty')
      if [[ $? -ne 0 || (-z "${HTTP_PROXY_URL}" && -z "$
{HTTPS_PROXY_URL}") ]]; then
        echo "Info: The config-json was parsed, but no proxy
settings were found."
        return 0
      fi

      check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}"
      check_protocol "${HTTPS_PROXY_URL}" "$
{supported_protocols[@]}"

      if ! grep -q 'http_proxy' /etc/environment; then
        echo "export http_proxy=${HTTP_PROXY_URL}
        export https_proxy=${HTTPS_PROXY_URL}
        export HTTP_PROXY=${HTTP_PROXY_URL}
        export HTTPS_PROXY=${HTTPS_PROXY_URL}
        export no_proxy=localhost,127.0.0.1" >> /etc/environment
        source /etc/environment
      fi

      # Configure Docker to use a proxy
      mkdir -p /etc/systemd/system/docker.service.d
      echo "[Service]
      Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\"
      Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\"
      Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/
system/docker.service.d/proxy.conf
      systemctl daemon-reload
      systemctl restart docker
```

Tabelle 3-5. Triton Inference Server-Container-Image (Fortsetzung)

Komponente	Beschreibung
	echo "Info: docker and system environment are now configured to use the proxy settings" } ■ Einzeiliges Bild im base64-Format codiert docker run -d --gpus all --rm -p8000:8000 -p8001:8001 -p8002:8002 -v /home/vmware/model_repository:/models nvcr.io/nvidia/tritonserver:*ngc_image_tag* tritonserver --model-repository=/models --model-control-mode=poll Geben Sie beispielsweise für „tritonserver:23.10-py3“ das folgende Skript im base64-Format an: ZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tcm0gLXA4MDAwOjgwMDAgLXA4MDAxOjgwMDEgLXA4MDAyOjgwMDIgLXYgL2hvbWUvdm13YXJlL21vZGVsX3JlcG9zaXRvcnk6L21vZGVscyBudmNyLmlvL252aWRpYS90cml0b25zZXJ2ZXI6MjMuMTAtcHkzIHRyaXRvbnNlcnZlciAtLW1vZGVsLXJlcG9zaXRvcnk9L21vZGVscyAtLW1vZGVsLWNvbnRyb2wtbW9kZT1wb2xs was dem folgenden Skript im Klartextformat entspricht: docker run -d --gpus all --rm -p8000:8000 -p8001:8001 -p8002:8002 -v /home/vmware/model_repository:/models nvcr.io/nvidia/tritonserver:23.10-py3 tritonserver --model-repository=/models --model-control-mode=poll ■ Geben Sie die Installationseigenschaften des vGPU-Gasttreibers ein, wie z. B. `vgpu-license` und `nvidia-portal-api-key`. ■ Geben Sie nach Bedarf Werte für die Eigenschaften an, die für eine getrennte Umgebung erforderlich sind. Weitere Informationen finden Sie unter OVF-Eigenschaften von Deep Learning-VMs.
Ausgabe	■ Installationsprotokolle für den vGPU-Gasttreiber in `/var/log/vgpu-install.log`. Um zu überprüfen, ob der vGPU-Gasttreiber installiert ist, melden Sie sich über SSH bei der VM an und führen Sie den Befehl `nvidia-smi` aus. ■ Cloud-init-Skriptprotokolle in `/var/log/dl.log`. ■ Triton Inference Server-Container. Um sicherzustellen, dass der Triton Inference Server-Container ausgeführt wird, führen Sie die Befehle `sudo docker ps -a` und `sudo docker logs container_id` aus. Das Modell-Repository für den Triton Inference Server befindet sich in `/home/vmware/model_repository`. Anfänglich ist das Modell-Repository leer, und das anfängliche Protokoll der Triton Inference Server-Instanz zeigt an, dass kein Modell geladen ist.

Erstellen eines Modell-Repository

Führen Sie die folgenden Schritte aus, um Ihr Modell für den Modell-Rückschluss zu laden:

1 Erstellen Sie das Modell-Repository für Ihr Modell.

Weitere Informationen finden Sie in der Dokumentation Dokumentation zum NVIDIA Triton Inference Server-Modell-Repository.

2 Kopieren Sie das Modell-Repository in `/home/vmware/model_repository`, damit der Triton Inference Server es laden kann.

```
sudo cp -r path_to_your_created_model_repository/* /home/vmware/model_repository/
```

Senden von Modell-Rückschlussanforderungen

1 Stellen Sie sicher, dass der Triton Inference Server fehlerfrei ist und die Modelle bereit sind, indem Sie diesen Befehl in der Deep Learning-VM-Konsole ausführen.

```
curl -v localhost:8000/v2/simple_sequence
```

2 Senden Sie eine Anforderung an das Modell, indem Sie diesen Befehl auf der Deep Learning-VM ausführen.

```
curl -v localhost:8000/v2/models/simple_sequence
```

Weitere Informationen zur Verwendung des Triton Inference-Servers finden Sie in der Dokumentation NVIDIA Triton Inference Server-Modell-Repository.

NVIDIA RAG

Sie können eine Deep Learning-VM verwenden, um RAG-Lösungen (Retrieval Augmented Generation) mit einem Llama2-Modell zu erstellen.

Weitere Informationen finden Sie in der Dokumentation zu NVIDIA RAG-Anwendungen mit Docker Compose (erfordert bestimmte Kontoberechtigungen).

Tabelle 3-6. NVIDIA RAG-Container-Image

<table>
<tr><th>Komponente</th><th>Beschreibung</th></tr>
<tr><td>Container-Images und -Modelle</td><td>

```
rag-app-text-chatbot.yaml
```

in der NVIDIA-Beispiel-RAG-Pipeline.
Informationen zu den für Deep Learning-VMs unterstützten NVIDIA RAG-Containeranwendungen finden Sie unter Versionshinweise zu VMware Deep Learning VM.</td></tr>
<tr><td>Erforderliche Eingaben</td><td>Um eine NVIDIA RAG-Arbeitslast bereitzustellen, müssen Sie die OVF-Eigenschaften für die Deep Learning-VMs wie folgt festlegen:

- Geben Sie ein cloud-init-Skript ein. Codieren Sie es im base64-Format.

  Geben Sie beispielsweise für Version 24.03 von NVIDIA RAG das folgende Skript an:


```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBwLnN
oCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCi
AgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0cmFwICdlc
nJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3JrbG9hZCInIEVS
UgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiCgogICAgY2F0IDw8RU9GID4gL29wdC9
kbHZtL2NvbmZpZy5qc29uCiAgICB7CiAgICAgICJfY29tbWVudCI6ICJUaGlzIHByb3ZpZG
VzIGRlZmF1bHQgc3VwcG9ydCBmb3IgUkFHOiBUZW5zb3JSVCBpbmZlcmVuY2UsIGxsYW1hM
i0xM2IgbW9kZWwsIGFuZCBIMTAweDIgR1BVIiwKICAgICAgInJhZyI6IHsKICAgICAgICAi
b3JnX25hbWUiOiAiY29jZndnYThqcTJjIiwKICAgICAgICAib3JnX3RlYW1fbmFtZSI6ICJ
uby10ZWFtIiwKICAgICAgICAicmFnX3JlcG9fbmFtZSI6ICJudmlkaWEvcGFpZiIsCiAgIC
AgICAgImxsbV9yZXBvX25hbWUiOiAibnZpZGlhL25pbSIsCiAgICAgICAgImVtYmVkX3Jlc
G9fbmFtZSI6ICJudmlkaWEvbmVtby1yZXRyaWV2ZXIiLAogICAgICAgICJyYWdfbmFtZSI6
ICJyYWctZG9ja2VyLWNvbXBvc2UiLAogICAgICAgICJyYWdfdmVyc2lvbiI6ICIyNC4wMyI
sCiAgICAgICAgImVtYmVkX25hbWUiOiAibnYtZW1iZWQtcWEiLAogICAgICAgICJlbWJlZF
90eXBlIjogIk5WLUVtYmVkLVFBIiwKICAgICAgICAiZW1iZWRfdmVyc2lvbiI6ICI0IiwKI
CAgICAgICAiaW5mZXJlbmNlX3R5cGUiOiAidHJ0IiwKICAgICAgICAibGxtX25hbWUiOiAi
bGxhbWEyLTEzYi1jaGF0IiwKICAgICAgICAibGxtX3ZlcnNpb24iOiAiaDEwMHgyX2ZwMTZ
fMjQuMDIiLAogICAgICAgICJudW1fZ3B1IjogIjIiLAogICAgICAgICJoZl90b2tlbiI6IC
JodWdnaW5nZmFjZSB0b2tlbiB0byBwdWxsIGxsbSBtb2RlbCwgdXBkYXRlIHdoZW4gdXNpb
mcgdmxsbSBpbmZlcmVuY2UiLAogICAgICAgICJoZl9yZXBvIjogImh1Z2dpbmdmYWNlIGxs
bSBtb2RlbCByZXBvc2l0b3J5LCB1cGRhdGUgd2hlbiB1c2luZyB2bGxtIGluZmVyZW5jZSI
KICAgICAgfQogICAgfQogICAgRU9GCiAgICBDT05GSUdfSlNPTj0kKGNhdCAiL29wdC9kbH
ZtL2NvbmZpZy5qc29uIikKICAgIElORkVSRU5DRV9UWVBFPSQoZWNobyAiJHtDT05GSUdfS
lNPTn0iIHwganEgLXIgJy5yYWcuaW5mZXJlbmNlX3R5cGUnKQogICAgaWYgWyAiJHtJTkZF
UkVOQ0VfVFlQRX0iID0gInRydCIgXTsgdGhlbgogICAgICByZXF1aXJlZF92YXJzPSgiT1J
HX05BTUUiICJPUkdfVEVBTV9OQU1FIiAiUkFHX1JFUE9fTkFNRSIgIkxMTV9SRVBPX05BTU
UiICJFTUJFRF9SRVBPX05BTUUiICJSQUdfTkFNRSIgIlJBR19WRVJTSU9OIiAiRU1CRURfT
kFNRSIgIkVNQkVEX1RZUEUiICJFTUJFRF9WRVJTSU9OIiAiTExNX05BTUUiICJMTE1fVkVS
U0lPTiIgIk5VTV9HUFUiKQogICAgZWxpZiBbICIke0lORkVSRU5DRV9UWVBFfSIgPSAidmx
sbSIgXTsgdGhlbgogICAgICByZXF1aXJlZF92YXJzPSgiT1JHX05BTUUiICJPUkdfVEVBTV
9OQU1FIiAiUkFHX1JFUE9fTkFNRSIgIkxMTV9SRVBPX05BTUUiICJFTUJFRF9SRVBPX05BT
UUiICJSQUdfTkFNRSIgIlJBR19WRVJTSU9OIiAiRU1CRURfTkFNRSIgIkVNQkVEX1RZUEUi
ICJFTUJFRF9WRVJTSU9OIiAiTExNX05BTUUiICJOVU1fR1BVIiAiSEZfVE9LRU4iICJIRl9
SRVBPIikKICAgIGVsc2UKICAgICAgZXJyb3JfZXhpdCAiSW5mZXJlbmNlIHR5cGUgJyR7SU
5GRVJFTkNFX1RZUEV9JyBpcyBub3QgcmVjb2duaXplZC4gTm8gYWN0aW9uIHdpbGwgYmUgd
GFrZW4uIgogICAgZmkKICAgIGZvciBpbmRleCBpbiAiJHshcmVxdWlyZWRfdmFycltAXX0i
OyBkbwogICAgICBrZXk9IiR7cmVxdWlyZWRfdmFyc1skaW5kZXhdfSIKICAgICAganFfcXV
lcnk9Ii5yYWcuJHtrZXksLH0gfCBzZWxlY3QgKC4hPW51bGwpIgogICAgICB2YWx1ZT0kKG
VjaG8gIiR7Q09ORklHX0pTT059IiB8IGpxIC1yICIke2pxX3F1ZXJ5fSIpCiAgICAgIGlmI
FtbIC16ICIke3ZhbHVlfSIgXV07IHRoZW4gCiAgICAgICAgZXJyb3JfZXhpdCAiJHtrZXl9
IGlzIHJlcXVpcmVkIGJ1dCBub3Qgc2V0LiIKICAgICAgZWxzZQogICAgICAgIGV2YWwgJHt
rZXl9PVwiIiR7dmFsdWV9IlwiCiAgICAgIGZpCiAgICBkb25lCgogICAgUkFHX1VSST0iJH
tSQUdfUkVQT19OQU1FfS8ke1JBR19OQU1FfToke1JBR19WRVJTSU9OfSIKICAgIEVNQkVEX
01PREVMX1VSST0iJHtFTUJFRF9SRVBPX05BTUV9LyR7RU1CRURfTkFNRX06JHtFTUJFRF9W
RVJTSU9OfSIKCiAgICBOR0NfQ0xJX1ZFUlNJT049IjMuNDEuMiIKICAgIE5HQ19DTElfVVJ
MPSJodHRwczovL2FwaS5uZ2MubnZpZGlhLmNvbS92Mi9yZXNvdXJjZXMvbnZpZGlhL25nYy
1hcHBzL25nY19jbGkvdmVyc2lvbnMvJHtOR0NfQ0xJX1ZFUlNJT059L2ZpbGVzL25nY2Nsa
```

</td></tr>
</table>

Tabelle 3-6. NVIDIA RAG-Container-Image (Fortsetzung)

Komponente	Beschreibung
	V9saW51eC56aXAiCgogICAgbWtkaXIgLXAgL29wdC9kYXRhCiAgICBjZCAvb3B0L2RhdGEK CiAgICBpZiBbICEgLWYgLmZpbGVfZG93bmxvYWRlZCBdOyB0aGVuCiAgICAgICMgY2xlYW4 gdXAKICAgICAgcm0gLXJmIGNvbXBvc2UuZW52ICR7UkFHX05BTUV9KiAke0xMTV9OQU1FfS ogbmdjKiAke0VNQkVEX05BTUV9KiAqLmpzb24gLmZpbGVfZG93bmxvYWRlZAoKICAgICAgI yBpbnN0YWxsIG5nYy1jbGkKICAgICAgd2dldCAtLWNvbnRlbnQtZGlzcG9zaXRpb24gJHtO R0NfQ0xJX1VSTH0gLU8gbmdjY2xpX2xpbnV4LnppcCAmJiB1bnppcCBuZ2NjbGlfbGludXg uemlwCiAgICAgIGV4cG9ydCBQQVRIPWBwd2RgL25nYy1jbGk6JHtQQVRIfQoKICAgICAgQV BJS0VZPSIiCiAgICAgIFJFR19VUkk9Im52Y3IuaW8iCgogICAgICBpZiBbWyAiJChncmVwI HJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2 YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpIiA9PSAqIiR7UkVHX1VSSX0iKiBdXTsgdGhlbgo gICAgICAgIEFQSUtFWT0kKGdyZXAgcmVnaXN0cnktcGFzc3dkIC9vcHQvZGx2bS9vdmYtZW 52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKICAgI CAgZmkKCiAgICAgICAgIFsgLXogIiR7QVBJS0VZfSIgXTsgdGhlbgogICAgICAgICAgZXJy b3JfZXhpdCAiTm8gQVBJS0VZIGZvdW5kIgogICAgICBmaQoKICAgICAgIyBjb25maWcgbmd jLWNsaQogICAgICBta2RpciAtcCB+Ly5uZ2MKCiAgICAgIGNhdCA8PCBFT0YgPiB+Ly5uZ2 MvY29uZmlnCiAgICAgIFtDVVJSRU5UXQogICAgICBhcGlrZXkgPSAke0FQSUtFWX0KICAgI CAgZm9ybWF0X3R5cGUgPSBhc2NpaQogICAgICBvcmcgPSAke09SR19OQU1FfQogICAgICB0 ZWFtID0gJHtPUkdfVEVBTV9OQU1FfQogICAgICBhY2UgPSBuby1hY2UKICAgICAgIEVPRgoKICA gICAgIyBuZ2MgZG9ja2VyIGxvZ2luCiAgICAgIGRvY2tlciBsb2dpbiBudmNyLmlvIC11IF wkb2F1dGh0b2tlbiAtcCAke0FQSUtFWX0KCiAgICAgICMgZG9ja2VyaHViIGxvZ2luIGZvc iBnZW5lcmFsIGNvbXBvbmVudHMsIGUuZy4gbWluaW8KICAgICAgRE9DS0VSSFVCX1VSST0k KGdyZXAgcmVnaXN0cnktMi11cmkgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICd zLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQogICAgICBET0NLRVJIVUJfVVNFUk 5BTUU9JChncmVwIHJlZ2lzdHJ5LTItdXNlciAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZ WQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCiAgICAgIERPQ0tFUkhV Ql9QQVNTV09SRD0kKGdyZXAgcmVnaXN0cnktMi1wYXNzd2QgL29wdC9kbHZtL292Zi1lbnY ueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIC AgaWYgW1sgLW4gIiR7RE9DS0VSSFVCX1VTRVJOQU1FfSIgJiYgLW4gIiR7RE9DS0VSSFVCX 1BBU1NXT1JEfSIgXV07IHRoZW4KICAgICAgICBkb2NrZXIgbG9naW4gLXUgJHtET0NLRVJI VUJfVVNFUk5BTUV9IC1wICR7RE9DS0VSSFVCX1BBU1NXT1JEfQogICAgICBlbHNlCiAgICA gICAgZWNobyAiV2FybmluZzogRG9ja2VySHViIG5vdCBsb2dpbiIKICAgICAgZmkKCiAgICC AgICMgZ2V0IFJBRyBmaWxlcwogICAgICBuZ2MgcmVnaXN0cnkgcmVzb3VyY2UgZG93bmxvY WQtdmVyc2lvbiAke1JBR19VUkl9CgogICAgICAjIGdldCBsbG0gbW9kZWwKICAgICAgaWYg WyAiJHtJTkZFUkVOQ0VfVFlQRX0iID0gInRydCIgXTsgdGhlbgogICAgICAgIExMTV9NT0R FTF9VUkk9IiR7TExNX1JFUE9fTkFNRX0vJHtMTE1fTkFNRX06JHtMTE1fVkVSU0lPTn0iCi AgICAgICAgbmdjIHJlZ2lzdHJ5IG1vZGVsIGRvd25sb2FkLXZlcnNpb24gJHtMTE1fTU9ER UxfVVJJfQogICAgICAgIGNobW9kIC1SIG8rclggJHtMTE1fTkFNRX1fdiR7TExNX1ZFUlNJ T059CiAgICAgICAgTExNX01PREVMX0ZPTERFUj0iL29wdC9kYXRhLyR7TExNX05BTUV9X3Y ke0xMTV9WRVJTSU9OfSIKICAgICAgZWxpZiBbICIke0lORkVSRU5DRV9UWVBFfSIgPSAidm xsbSIgXTsgdGhlbgogICAgICAgIHBpcCBpbnN0YWxsIGh1Z2dpbmdmYWNlX2h1YgogICAgI CAgIGh1Z2dpbmdmYWNlLWNsaSBsb2dpbiAtLXRva2VuICR7SEZfVE9LRU59CiAgICAgICAg aHVnZ2luZ2ZhY2UtY2xpIGRvd25sb2FkIC0tcmVzdW1lLWRvd25sb2FkICR7SEZfUkVQT30 vJHtMTE1fTkFNRX0gLS1sb2NhbC1kaXIgJHtMTE1fTkFNRX0gLS1sb2NhbC1kaXItdXNlLX N5bWxpbmtzIEZhbHNlCiAgICAgICAgTExNX01PREVMX0ZPTERFUj0iL29wdC9kYXRhLyR7T ExNX05BTUV9IgogICAgICAgIGNhdCA8PCBFT0YgPiAke0xMTV9NT0RFTF9GT0xERVJ9L21v ZGVsX2NvbmZpZy55YW1sIAogICAgICAgIGVuZ2luZToKICAgICAgICAgIG1vZGVsOiAvbW9 kZWwtc3RvcmUKICAgICAgICAgIGVuZm9yY2VfZWFnZXI6IGZhbHNlCiAgICAgICAgICBtYX hfY29udGV4dF9sZW5fdG9fY2FwdHVyZTogODE5MgogICAgICAgICAgICAgIWF4X251bV9zZXFzO iAyNTYKICAgICAgICAgICAgIGR0eXBlOiBmbG9hdDE2CiAgICAgICAgICAgICB0ZW5zb3JfcGFyYWxs ZWxfc2l6ZTogJHtOVU1fR1BVfQogICAgICAgICAgICAgIGVuZ2luZToKICAgICAgICAgICAgIG1vZGVsOiAvbW9 kZWwtc3RvcmUKICAgICAgICAgICAgIGVuZm9yY2VfZWFnZXI6IGZhbHNlCiAgICAgICAgICBtYX hfY29udGV4dF9sZW5fdG9fY2FwdHVyZTogODE5MgogICAgICAgICAgICAgIWF4X251bV9zZXFzO iAyNTYKICAgICAgICAgICAgIGR0eXBlOiBmbG9hdDE2CiAgICAgICAgICAgICB0ZW5zb3JfcGFyYWxs ZWxfc2l6ZTogJHtOVU1fR1BVfQogICAgICAgICAgICAgIGVuZ2luZToKICAgICAgICAgICAgIG1vZGVsOiAvbW9 gMC44CiAgICBFT0YKICAgICAgICBjaG1vZCAtUiBvK3JYICR7TExNX01PREVMX0ZPTERFUn 0KICAgICAgICBweXRob24zIC1jICJpbXBvcnQgeWFtbCwganNvbiwgc3lzOyBwcmludChqc 29uLmR1bXBzKHlhbWwuc2FmZV9sb2FkKHN5cy5zdGRpbi5yZWFkKCkpKSkiIDwgIiR7UkFH X05BTUV9X3Yke1JBR19WRVJTSU9OfS9yYWctYXBwLXRleHQtY2hhdGJvdC55YW1sIj4gcmF nLWFwcC10ZXh0LWNoYXRib3QuanNvbgogICAgICAgICAgICAgIGpxICcuc2Vydmljc2XMuIm5lbW9sbG 0taW5mZXJlbmNlIi5pbWFnZSA9ICJudmNyLmlvL252aWRpYS9uaW0vbmltX2xsbToyNC4wM i1kYXkwIiB8CiAgICAgICAgICAgICAgICAgIC5zZXJ2aWNlcy4ibmVtb2xsbS1pbmZlcmVuY2UiLmNv bW1hbmQgPSAibltX3ZsbG0gLS1tb2RlbF9uYW1lICR7TU9ERUxfTkFNRX0gLS1tb2RlbF9 jb25maWcgL21vZGVsLXN0b3JlL21vZGVsX2NvbmZpZy55YW1sIiB8CiAgICAgICAgICAgICAgICAgIC 5zZXJ2aWNlcy4ibmVtb2xsbS1pbmZlcmVuY2UiLnBvcnRzICs9IFsiODAwMDo4MDAwIl0gf AogICAgICAgICAgICAgICAgICAuc2Vydmljc2XMuIm5lbW9sbG0taW5mZXJlbmNlIi5leHBvc2UgKz0g

Tabelle 3-6. NVIDIA RAG-Container-Image (Fortsetzung)

Komponente	Beschreibung
	WyI4MDAwIl0nIHJhZy1hcHAtdGV4dC1jaGF0Ym90Lmpzb24gPiB0ZW1wLmpzb24gJiYgbXY gdGVtcC5qc29uIHJhZy1hcHAtdGV4dC1jaGF0Ym90Lmpzb24KICAgICAgICBweXRob24zIC 1jICJpbXBvcnQgeWFtbCwganNvbiwgc3lzOyBwcmludCh5YW1sLnNhZmVfZHVtcChqc29uL mxvYWQoc3lzLnN0ZGluKSwgZGVmYXVsdF9mbG93X3N0eWxlPUZhbHNlLCBzb3J0X2tleXM9 RmFsc2UpKSIgPCByYWctYXBwLXRleHQtY2hhdGJvdC5qc29uID4gIiR7UkFHX05BTUV9X3Y ke1JBR19WRVJTSU9OfS9yYWctYXBwLXRleHQtY2hhdGJvdC55YW1sIgogICAgICBmaQoKIC AgICAgIyBnZXQgZW1iZWRkaW5nIG1vZGVscwogICAgICBuZ2MgcmVnaXN0cnkgbW9kZWwgZ G93bmxvYWQtdmVyc2lvbiAke0VNQkVEX01PREVMX1VSSX0KICAgICAgY2htb2QgLVIgbyty WCAke0VNQkVEX05BTUV9X3Yke0VNQkVEX1ZFUlNJT059CgogICAgICAjIGNvbmZpZyBjb21 wb3NlLmVudgogICAgICBjYXQgPDwgRU9GID4gY29tcG9zZS5lbnYKICAgICAgZXhwb3J0IE 1PREVMX0RJUkVDVE9SWT0iJHtMTE1fTU9ERUxfRk9MREVSfSIKICAgICAgZXhwb3J0IE1PR EVMX05BTUU9JHtMTE1fTkFNRX0KICAgICAgZXhwb3J0IE5VTV9HUFU9JHtOVU1fR1BVfQog ICAgICBleHBvcnQgQVBQX0NPTkZJR19GSUxFPS9kZXYvbnVsbAogICAgICBleHBvcnQgRU1 CRURESU5HX01PREVMX0RJUkVDVE9SWT0iL29wdC9kYXRhLyR7U1CRURfTkFNRX1fdiR7RU 1CRURfVkVSU0lPTn0iCiAgICAgIGV4cG9ydCBFTUJFRERJTkdfTU9ERUxfTkFNRT0ke0VNQ kVEX1RZUEV9CiAgICAgIGV4cG9ydCBFTUJFRERJTkdfTU9ERUxfQ0tQVF9OQU1FPSIke0VN QkVEX1RZUEV9LSR7RU1CRURfVkVSU0lPTn0ubmVtbyIKICAgIEVPRgoKICAgICAgdG91Y2g gLmZpbGVfZG93bmxvYWRlZAogICAgZmkKCiAgICAgIHN0YXJ0IE5HQyBSQUcKICAgIGRvY2 tlciBjb21wb3NlIC1mICR7UkFHX05BTUV9X3Yke1JBR19WRVJTSU9OfS9kb2NrZXItY29tc G9zZS12ZWN0b3JkYi55YW1sIHVwIC1kIHBndmVjdG9yCiAgICBzb3VyY2UgY29tcG9zZS5l bnY7IGRvY2tlciBjb21wb3NlIC1mICR7UkFHX05BTUV9X3Yke1JBR19WRVJTSU9OfS9yYWc tYXBwLXRleHQtY2hhdGJvdC55YW1sIHVwIC1kCgotIHBhdGg6IC9vcHQvZGx2bS91dGlscy 5zaAogIHBlcm1pc3Npb25zOiAnMDc1NScKICBjb250ZW50OiB8CiAgICAjIS9iaW4vYmFza AogICAgZXJyb3JfZXhpdCgpIHsKICAgICAgZWNobyAiRXJyb3I6ICQxIiA+JjIKICAgICAg dm10b29sc2QgLS1jbWQgImluZm8tc2V0IGd1ZXN0aW5mby52bXNlcnZpY2UuYm9vdHN0cmF wLmNvbmRpdGlvbiBmYWxzZSwgREExXb3JrbG9hZEZhaWx1cmUsICQxIgogICAgICBleGl0ID EKICAgIH0KCiAgICBjaGVja19wcm90b2NvbCgpIHsKICAgICAgbG9jYWwgcHJveHlfdXJsP SQxCiAgICAgIHNoaWZ0CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIp CiAgICAgIGlmIFtbIC1uICIke3Byb3h5X3VybH0iIF1dOyB0aGVuCiAgICAgICAgbG9jYWw gcHJvdG9jb2w9JChlY2hvICIke3Byb3h5X3VybH0iIHwgYXdrIC1GICc6Ly8nICd7aWYgKE 5GID4gMSkgcHJpbnQgJDE7IGVsc2UgcHJpbnQgIiJ9JykKICAgICAgICBpZiBbIC16ICIkc HJvdG9jb2wiIF07IHRoZW4KICAgICAgICAgIGVjaG8gIk5vIHNwZWNpZmljIHByb3RvY29s IHByb3ZpZGVkLiBTa2lwcGluZyBwcm90b2NvbCBjaGVjay4iCiAgICAgICAgICByZXR1cm4 gMAogICAgICAgIGZpCiAgICAgICAgbG9jYWwgcHJvdG9jb2xfaW5jbHVkZWQ9ZmFsc2UKIC AgICAgICBmb3IgdmFyIGluICIke3N1cHBvcnRlZF9wcm90b2NvbHNbQF19IjsgZG8KICAgI CAgICAgIGlmIFtbICIke3Byb3RvY29sfSIgPT0gIiR7dmFyfSIgXV07IHRoZW4KICAgICAg ICAgICAgcHJvdG9jb2xfaW5jbHVkZWQ9dHJ1ZQogICAgICAgICAgICBicmVhawogICAgICA gICAgZmkKICAgICAgICBkb25lCiAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2xfaW5jbHVkZW R9IiA9PSBmYWxzZSBdXTsgdGhlbgogICAgICAgICAgZXJyb3JfZXhpdCAiVW5zdXBwb3J0Z WQgcHJvdG9jb2w6ICR7cHJvdG9jb2x9LiBTdXBwb3J0ZWQgcHJvdG9jb2xzIGFyZTogJHtz dXBwb3J0ZWRfcHJvdG9jb2xzWypdfSIKICAgICAgICAgICBmaQogICAgICBmaQogICAgfQoKICA gICMgJEA6IGxpc3Qgb2Ygc3VwcG9ydGVkIHByb3RvY29scwogICAgc2V0X3Byb3h5KCkgew ogICAgICBsb2NhbCBzdXBwb3J0ZWRfcHJvdG9jb2xzPSgiJEAiKQoKICAgICAgQ09ORklHX 0pTT05fQkFTRTY0PSQoZ3JlcCAnY29uZmlnLWpzb24nIC9vcHQvZGx2bS9vdmYtZW52Lnht bCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKICAgICAgQ09 ORklHX0pTT049JChlY2hvICR7Q09ORklHX0pTT05fQkFTRTY0fSB8IGJhc2U2NCAtLWRlY2 9kZSkKCiAgICAgIEhUVFBfUFJPWFlfVVJMPSQoZWNobyAiJHtDT05GSUdfSlNPTn0iIHwga nEgLXIgJy5odHRwX3Byb3h5IC8vIGVtcHR5JykKICAgICAgICAgSFRUUFNfUFJPWFlfVVJMPSQo ZWNobyAiJHtDT05GSUdfSlNPTn0iIHwganEgLXIgJy5odHRwc19wcm94eSAvLyBlbXB0eSc pCiAgICAgIGlmIFtbICQ/ IC1uZSAwIHx8ICgteiAiJHtIVFRQX1BST1hZX1VSTH0iICYmIC16ICIke0hUVFBTX1BST1h ZX1VSTH0iKSBdXTsgdGhlbgogICAgICAgIGluZm86IFRoZSBjb25maWctctanNvbi B3YXMgcGFyc2VkLCBidXQgbm8gcHJveHkgc2V0dGluZ3Mgd2VyZSBmb3VuZC4iCiAgICAgI CAgcmV0dXJuIDAKICAgICAgZmkKCiAgICAgIGNoZWNrX3Byb3RvY29sICIke0hUVFBfUFJP WFlfVVJMfSIgIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iCiAgICAgIGNoZWNrX3Byb3R vY29sICIke0hUVFBTX1BST1hZX1VSTH0iICIke3N1cHBvcnRlZF9wcm90b2NvbHNbQF19Ig oKICAgICAgaWYgISBncmVwIC1xICdodHRwX3Byb3h5JyAvZXRjL2Vudmlyb25tZW50OyB0a GVuCiAgICAgICAgZWNobyAiZXhwb3J0IGh0dHBfcHJveHk9JHtIVFRQX1BST1hZX1VSTH0K ICAgICAgICBleHBvcnQgaHR0cHNfcHJveHk9JHtIVFRQU19QUk9YWV9VUkx9CiAgICAgICA gZXhwb3J0IEhUVFBfUFJPWFk9JHtIVFRQX1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSF RUUFNfUFJPWFk9JHtIVFRQU19QUk9YWV9VUkx9CiAgICAgICAgZXhwb3J0IG5vX3Byb3h5P

Tabelle 3-6. NVIDIA RAG-Container-Image (Fortsetzung)

Komponente	Beschreibung


```
WxvY2FsaG9zdCwxMjcuMC4wLjEiID4+IC9ldGMvZW52aXJvbm1lbnQKICAgICAgICBzb3Vy
Y2UgL2V0Yy9lbnZpcm9ubWVudAogICAgICBmaQogICAgICAKICAgICAgIyBDb25maWd1cmU
gRG9ja2VyIHRvIHVzZSBhIHByb3h5CiAgICAgIG1rZGlyIC1wIC9ldGMvc3lzdGVtZC9zeX
N0ZW0vZG9ja2VyLnNlcnZpY2UuZAogICAgICBlY2hvICJbU2VydmljZV0KICAgICAgRW52a
XJvbm1lbnQ9XCJIVFRQX1BST1hZPSR7SFRUUF9QUk9YWV9VUkx9XCIKICAgICAgRW52aXJv
bm1lbnQ9XCJIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9
ubWVudD1cIk5PX1BST1hZPWxvY2FsaG9zdCwxMjcuMC4wLjFcIiIgPiAvZXRjL3N5c3RlbW
Qvc3lzdGVtL2RvY2tlci5zZXJ2aWNlLmQvcHJveHkuY29uZgogICAgICBzeXN0ZW1jdGwgZ
GFlbW9uLXJlbG9hZAogICAgICBzeXN0ZW1jdGwgcmVzdGFydCBkb2NrZXIKCiAgICAgIGVj
aG8gIkluZm86IGRvY2tlciBhbmQgc3lzdGVtIGVudmlyb25tZW50IGFyZSBub3cgY29uZml
ndXJlZCB0byB1c2UgdGhlIHByb3h5IHNldHRpbmdzIgogICAgfQ==
```


was dem folgenden Skript im Klartextformat entspricht:

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    set -eu
    source /opt/dlvm/utils.sh
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    set_proxy "http" "https"

    cat <<EOF > /opt/dlvm/config.json
    {
      "_comment": "This provides default support for RAG: TensorRT
inference, llama2-13b model, and H100x2 GPU",
      "rag": {
        "org_name": "cocfwga8jq2c",
        "org_team_name": "no-team",
        "rag_repo_name": "nvidia/paif",
        "llm_repo_name": "nvidia/nim",
        "embed_repo_name": "nvidia/nemo-retriever",
        "rag_name": "rag-docker-compose",
        "rag_version": "24.03",
        "embed_name": "nv-embed-qa",
        "embed_type": "NV-Embed-QA",
        "embed_version": "4",
        "inference_type": "trt",
        "llm_name": "llama2-13b-chat",
        "llm_version": "h100x2_fp16_24.02",
        "num_gpu": "2",
        "hf_token": "huggingface token to pull llm model, update when
using vllm inference",
        "hf_repo": "huggingface llm model repository, update when
using vllm inference"
      }
    }
    EOF
    CONFIG_JSON=$(cat "/opt/dlvm/config.json")
    INFERENCE_TYPE=$(echo "${CONFIG_JSON}" | jq -r
'.rag.inference_type')
    if [ "${INFERENCE_TYPE}" = "trt" ]; then
      required_vars=("ORG_NAME" "ORG_TEAM_NAME" "RAG_REPO_NAME"
"LLM_REPO_NAME" "EMBED_REPO_NAME" "RAG_NAME" "RAG_VERSION"
"EMBED_NAME" "EMBED_TYPE" "EMBED_VERSION" "LLM_NAME" "LLM_VERSION"
"NUM_GPU")
    elif [ "${INFERENCE_TYPE}" = "vllm" ]; then
```

Tabelle 3-6. NVIDIA RAG-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
      required_vars=("ORG_NAME" "ORG_TEAM_NAME" "RAG_REPO_NAME"
"LLM_REPO_NAME" "EMBED_REPO_NAME" "RAG_NAME" "RAG_VERSION"
"EMBED_NAME" "EMBED_TYPE" "EMBED_VERSION" "LLM_NAME" "NUM_GPU"
"HF_TOKEN" "HF_REPO")
    else
      error_exit "Inference type '${INFERENCE_TYPE}' is not
recognized. No action will be taken."
    fi
    for index in "${!required_vars[@]}"; do
      key="${required_vars[$index]}"
      jq_query=".rag.${key,,} | select (.!=null)"
      value=$(echo "${CONFIG_JSON}" | jq -r "${jq_query}")
      if [[ -z "${value}" ]]; then
        error_exit "${key} is required but not set."
      else
        eval ${key}=\""${value}"\"
      fi
    done

    RAG_URI="${RAG_REPO_NAME}/${RAG_NAME}:${RAG_VERSION}"
    EMBED_MODEL_URI="${EMBED_REPO_NAME}/${EMBED_NAME}:${EMBED_VERSION}"

    NGC_CLI_VERSION="3.41.2"
    NGC_CLI_URL="https://api.ngc.nvidia.com/v2/resources/nvidia/ngc-
apps/ngc_cli/versions/${NGC_CLI_VERSION}/files/ngccli_linux.zip"

    mkdir -p /opt/data
    cd /opt/data

    if [ ! -f .file_downloaded ]; then
      # clean up
      rm -rf compose.env ${RAG_NAME}* ${LLM_NAME}* ngc* ${EMBED_NAME}*
*.json .file_downloaded

      # install ngc-cli
      wget --content-disposition ${NGC_CLI_URL} -O ngccli_linux.zip &&
unzip ngccli_linux.zip
      export PATH=`pwd`/ngc-cli:${PATH}

      APIKEY=""
      REG_URI="nvcr.io"

      if [[ "$(grep registry-uri /opt/dlvm/ovf-env.xml | sed -n
's/.*oe:value="\([^"]*\).*/\1/p')" == *"${REG_URI}"* ]]; then
        APIKEY=$(grep registry-passwd /opt/dlvm/ovf-env.xml | sed -n
's/.*oe:value="\([^"]*\).*/\1/p')
      fi

      if [ -z "${APIKEY}" ]; then
          error_exit "No APIKEY found"
      fi

      # config ngc-cli
      mkdir -p ~/.ngc

      cat << EOF > ~/.ngc/config
      [CURRENT]
      apikey = ${APIKEY}
      format_type = ascii
      org = ${ORG_NAME}
      team = ${ORG_TEAM_NAME}
```

Tabelle 3-6. NVIDIA RAG-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
      ace = no-ace
    EOF

      # ngc docker login
      docker login nvcr.io -u \$oauthtoken -p ${APIKEY}

      # dockerhub login for general components, e.g. minio
      DOCKERHUB_URI=$(grep registry-2-uri /opt/dlvm/ovf-env.xml | sed
-n 's/.*oe:value="\([^"]*\).*/\1/p')
      DOCKERHUB_USERNAME=$(grep registry-2-user /opt/dlvm/ovf-env.xml
| sed -n 's/.*oe:value="\([^"]*\).*/\1/p')
      DOCKERHUB_PASSWORD=$(grep registry-2-passwd /opt/dlvm/ovf-
env.xml | sed -n 's/.*oe:value="\([^"]*\).*/\1/p')

      if [[ -n "${DOCKERHUB_USERNAME}" && -n "$
{DOCKERHUB_PASSWORD}" ]]; then
        docker login -u ${DOCKERHUB_USERNAME} -p ${DOCKERHUB_PASSWORD}
      else
        echo "Warning: DockerHub not login"
      fi

      # get RAG files
      ngc registry resource download-version ${RAG_URI}

      # get llm model
      if [ "${INFERENCE_TYPE}" = "trt" ]; then
        LLM_MODEL_URI="${LLM_REPO_NAME}/${LLM_NAME}:${LLM_VERSION}"
        ngc registry model download-version ${LLM_MODEL_URI}
        chmod -R o+rX ${LLM_NAME}_v${LLM_VERSION}
        LLM_MODEL_FOLDER="/opt/data/${LLM_NAME}_v${LLM_VERSION}"
      elif [ "${INFERENCE_TYPE}" = "vllm" ]; then
        pip install huggingface_hub
        huggingface-cli login --token ${HF_TOKEN}
        huggingface-cli download --resume-download ${HF_REPO}/$
{LLM_NAME} --local-dir ${LLM_NAME} --local-dir-use-symlinks False
        LLM_MODEL_FOLDER="/opt/data/${LLM_NAME}"
        cat << EOF > ${LLM_MODEL_FOLDER}/model_config.yaml
        engine:
          model: /model-store
          enforce_eager: false
          max_context_len_to_capture: 8192
          max_num_seqs: 256
          dtype: float16
          tensor_parallel_size: ${NUM_GPU}
          gpu_memory_utilization: 0.8
    EOF
        chmod -R o+rX ${LLM_MODEL_FOLDER}
        python3 -c "import yaml, json, sys;
print(json.dumps(yaml.safe_load(sys.stdin.read())))" < "${RAG_NAME}_v$
{RAG_VERSION}/rag-app-text-chatbot.yaml"> rag-app-text-chatbot.json
        jq '.services."nemollm-inference".image = "nvcr.io/nvidia/nim/
nim_llm:24.02-day0" |
            .services."nemollm-inference".command = "nim_vllm
--model_name ${MODEL_NAME} --model_config /model-store/
model_config.yaml" |
            .services."nemollm-inference".ports += ["8000:8000"] |
            .services."nemollm-inference".expose += ["8000"]' rag-app-
text-chatbot.json > temp.json && mv temp.json rag-app-text-chatbot.json
        python3 -c "import yaml, json, sys;
print(yaml.safe_dump(json.load(sys.stdin), default_flow_style=False,
sort_keys=False))" < rag-app-text-chatbot.json > "${RAG_NAME}_v$
```

Tabelle 3-6. NVIDIA RAG-Container-Image (Fortsetzung)

Komponente	Beschreibung

```
{RAG_VERSION}/rag-app-text-chatbot.yaml"
      fi

      # get embedding models
      ngc registry model download-version ${EMBED_MODEL_URI}
      chmod -R o+rX ${EMBED_NAME}_v${EMBED_VERSION}

      # config compose.env
      cat << EOF > compose.env
      export MODEL_DIRECTORY="${LLM_MODEL_FOLDER}"
      export MODEL_NAME=${LLM_NAME}
      export NUM_GPU=${NUM_GPU}
      export APP_CONFIG_FILE=/dev/null
      export EMBEDDING_MODEL_DIRECTORY="/opt/data/${EMBED_NAME}_v$
{EMBED_VERSION}"
      export EMBEDDING_MODEL_NAME=${EMBED_TYPE}
      export EMBEDDING_MODEL_CKPT_NAME="${EMBED_TYPE}-$
{EMBED_VERSION}.nemo"
    EOF

      touch .file_downloaded
    fi

    # start NGC RAG
    docker compose -f ${RAG_NAME}_v${RAG_VERSION}/docker-compose-
vectordb.yaml up -d pgvector
    source compose.env; docker compose -f ${RAG_NAME}_v${RAG_VERSION}/
rag-app-text-chatbot.yaml up -d

- path: /opt/dlvm/utils.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    error_exit() {
      echo "Error: $1" >&2
      vmtoolsd --cmd "info-set guestinfo.vmservice.bootstrap.condition
false, DLWorkloadFailure, $1"
      exit 1
    }

    check_protocol() {
      local proxy_url=$1
      shift
      local supported_protocols=("$@")
      if [[ -n "${proxy_url}" ]]; then
        local protocol=$(echo "${proxy_url}" | awk -F '://' '{if (NF >
1) print $1; else print ""}')
        if [ -z "$protocol" ]; then
          echo "No specific protocol provided. Skipping protocol
check."
          return 0
        fi
        local protocol_included=false
        for var in "${supported_protocols[@]}"; do
          if [[ "${protocol}" == "${var}" ]]; then
            protocol_included=true
            break
          fi
        done
        if [[ "${protocol_included}" == false ]]; then
          error_exit "Unsupported protocol: ${protocol}. Supported
```

Tabelle 3-6. NVIDIA RAG-Container-Image (Fortsetzung)

Komponente	Beschreibung
	protocols are: ${supported_protocols[*]}" fi fi } # $@: list of supported protocols set_proxy() { local supported_protocols=("$@") CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml \| sed -n 's/.*oe:value="\([^"]*\).*/\1/p') CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} \| base64 --decode) HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" \| jq -r '.http_proxy // empty') HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r '.https_proxy // empty') if [[$? -ne 0 \|\| (-z "${HTTP_PROXY_URL}" && -z "${HTTPS_PROXY_URL}")]]; then echo "Info: The config-json was parsed, but no proxy settings were found." return 0 fi check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}" check_protocol "${HTTPS_PROXY_URL}" "${supported_protocols[@]}" if ! grep -q 'http_proxy' /etc/environment; then echo "export http_proxy=${HTTP_PROXY_URL} export https_proxy=${HTTPS_PROXY_URL} export HTTP_PROXY=${HTTP_PROXY_URL} export HTTPS_PROXY=${HTTPS_PROXY_URL} export no_proxy=localhost,127.0.0.1" >> /etc/environment source /etc/environment fi # Configure Docker to use a proxy mkdir -p /etc/systemd/system/docker.service.d echo "[Service] Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\" Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\" Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/system/docker.service.d/proxy.conf systemctl daemon-reload systemctl restart docker echo "Info: docker and system environment are now configured to use the proxy settings" } ■ Geben Sie die Installationseigenschaften des vGPU-Gasttreibers ein, wie z. B. `vgpu-license` und `nvidia-portal-api-key`. ■ Geben Sie nach Bedarf Werte für die Eigenschaften an, die für eine getrennte Umgebung erforderlich sind. Weitere Informationen finden Sie unter OVF-Eigenschaften von Deep Learning-VMs.
Ausgabe	■ Installationsprotokolle für den vGPU-Gasttreiber in `/var/log/vgpu-install.log`. Um zu überprüfen, ob der vGPU-Gasttreiber installiert ist, melden Sie sich über SSH bei der VM an und führen Sie den Befehl `nvidia-smi` aus.

Tabelle 3-6. NVIDIA RAG-Container-Image (Fortsetzung)

Komponente	Beschreibung
	■ Cloud-init-Skriptprotokolle in `/var/log/dl.log`. Um den Fortschritt der Bereitstellung zu verfolgen, führen Sie `tail -f /var/log/dl.log` aus. ■ Beispiel einer Chatbot-Webanwendung, auf die Sie unter `http://dl_vm_ip:3001/orgs/nvidia/models/text-qa-chatbot` zugreifen können Sie können Ihre eigene Knowledgebase hochladen.

Zuweisen einer statischen IP-Adresse zu einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA

Standardmäßig werden die Deep Learning-VM-Images auf Zuweisung von Adressen mittels DHCP konfiguriert. Wenn Sie eine Deep Learning-VM mit einer statischen IP-Adresse direkt auf einem vSphere-Cluster bereitstellen möchten, müssen Sie dem Abschnitt „cloud-init“ zusätzlichen Code hinzufügen.

Auf vSphere with Tanzu wird die Zuweisung von IP-Adressen durch die Netzwerkkonfiguration für den Supervisor in NSX bestimmt.

Verfahren

1 Erstellen Sie ein cloud-init-Skript im Klartextformat für die DL-Arbeitslast, die Sie verwenden möchten.

Weitere Informationen finden Sie unter Deep Learning-Arbeitslasten in VMware Private AI Foundation with NVIDIA.

2 Fügen Sie dem cloud-init-Skript den folgenden zusätzlichen Code hinzu.

```
#cloud-config
<instructions_for_your_DL_workload>

manage_etc_hosts: true

write_files:
  - path: /etc/netplan/50-cloud-init.yaml
    permissions: '0600'
    content: |
      network:
        version: 2
        renderer: networkd
        ethernets:
          ens33:
            dhcp4: false # disable DHCP4
            addresses: [x.x.x.x/x]  # Set the static IP address and mask
            routes:
                - to: default
                  via: x.x.x.x # Configure gateway
            nameservers:
              addresses: [x.x.x.x, x.x.x.x] # Provide the DNS server address. Separate
```

```
mulitple DNS server addresses with commas.

runcmd:
  - netplan apply
```

3 Codieren Sie das resultierende cloud-init-Skript im base64-Format.

4 Legen Sie das resultierende cloud-init-Skript im base64-Format als Wert auf den `user-data`-OVF-Parameter des Deep Learning-VM-Images fest.

Beispiel: Zuweisen einer statischen IP-Adresse zu einer CUDA-Beispielarbeitslast

Für ein Beispiel für eine Deep Learning-VM mit einer Deep Learning-Arbeitslasten in VMware Private AI Foundation with NVIDIA-DL-Arbeitslast:

Deep Learning-VM-Element	Beispielwert
DL-Arbeitslast-Image	nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8
IP-Adresse	10.199.118.245
Subnetzpräfix	/25
Gateway	10.199.118.253
DNS-Server	■ 10.142.7.1 ■ 10.132.7.1

Stellen Sie den folgenden cloud-init-Code bereit:

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBwLnNoCiAgcGVybWlzc2lvbnM6IC
cwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBkb2NrZXIgcnVuIC1kIG52Y3IuaW8vbnZpZGlhL2s4
cy9jdWRhLXNhbXBsZTp2ZWN0b3JhZGQtY3VkYTExLjcuMS11Ymk4CgptYW5hZ2VfZXRjX2hvc3RzOiB0cnVlCiAKd3JpdG
VfZmlsZXM6CiAgLSBwYXRoOiAvZXRjL25ldHBsYW4vNTAtY2xvdWQtaW5pdC55YW1sCiAgICBwZXJtaXNzaW9uczogJzA2
MDAnCiAgICBjb250ZW50OiB8CiAgICAgIG5ldHdvcms6CiAgICAgICAgdmVyc2lvbjogMgogICAgICAgIHJlbmRlcmVyOi
BuZXR3b3JrZAogICAgICAgIGV0aGVybmV0czoKICAgICAgICAgIGVuczMzOgogICAgICAgICAgICBkaGNwNDogZmFsc2Ug
IyBkaXNhYmxlIERIQ1A0CiAgICAgICAgICAgIGFkZHJlc3NlczogWzEwLjE5OS4xMTguMjQ1LzI1XSAgIyBTZXQgdGhlIH
N0YXRpYyBJUCBhZGRyZXNzIGFuZCBtYXNrCiAgICAgICAgICAgIHJvdXRlczoKICAgICAgICAgICAgICAtIHRvOiAxMC4w
ZmF1bHQKICAgICAgICAgICAgICAgIHZpYTogMTE4LjI1MyAjIENvbmZpZ3VyZSBnYXRld2F5CiAgICAgICAgICAgICAgIC
AgICAgICAgIG5hbWVzZXJ2ZXJzOgogICAgICAgICAgICAgIGFkZHJlc3NlczogWzEwLjE0Mi43LjEsIDEwLjEzMi43LjFdICMg
UHJvdmlkZSB0aGUgRE5TIHNlcnZlciBhZGRyZXNzLiBTZXBhcmF0ZSBtdWxpdHBsZSBETlMgc2VydmVyIGFkZHJlc3Nlcy
B3aXRoIGNvbW1hcy4KIApydW5jbWQ6CiAgLSBuZXRwbGFuIGFwcGx5
```

was dem folgenden Skript im Klartextformat entspricht:

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    docker run -d nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8

manage_etc_hosts: true
```

```
write_files:
  - path: /etc/netplan/50-cloud-init.yaml
    permissions: '0600'
    content: |
      network:
        version: 2
        renderer: networkd
        ethernets:
          ens33:
            dhcp4: false # disable DHCP4
            addresses: [10.199.118.245/25]  # Set the static IP address and mask
            routes:
                - to: default
                  via: 10.199.118.253 # Configure gateway
            nameservers:
              addresses: [10.142.7.1, 10.132.7.1] # Provide the DNS server address. Separate
mulitple DNS server addresses with commas.

runcmd:
  - netplan apply
```

Bereitstellen einer Deep Learning-VM mit einer Proxyserver

Zum Herstellen einer Verbindung zwischen der Deep Learning-VM und dem Internet in einer getrennten Umgebung, in der über einen Proxyserver auf das Internet zugegriffen wird, müssen Sie die Details des Proxyservers in der Datei `config.json` auf der virtuellen Maschine bereitstellen.

Verfahren

1 Erstellen Sie eine JSON-Datei mit den Eigenschaften für den Proxyserver.

Proxyserver, der keine Authentifizierung benötigt	`{` `  "http_proxy": "`*protocol*`://`*ip-address-or-fqdn*`:`*port*`",` `  "https_proxy": "`*protocol*`://`*ip-address-or-fqdn*`:`*port*`"` `}`
Proxyserver, der Authentifizierung benötigt	`{` `  "http_proxy": "`*protocol*`://`*username*`:`*password*`@`*ip-address-or-fqdn*`:`*port*`",` `  "https_proxy": "`*protocol*`://`*username*`:`*password*`@`*ip-address-or-fqdn*`:`*port*`"` `}`

wobei:

- *protocol* als das vom Proxyserver verwendete Protokoll fungiert, wie z. B. `http` oder `https`.

- *username* und *password* als Anmeldedaten für die Authentifizierung beim Proxyserver fungieren. Wenn der Proxyserver keine Authentifizierung benötigt, überspringen Sie diese Parameter.
- *ip-address-or-fqdn*: Die IP-Adresse oder der Hostname des Proxyservers.
- *port*: Die Portnummer, für die der Proxyserver eingehende Anforderungen überwacht.

2 Kodieren Sie den JSON-Ergebniscode im base64-Format.

3 Wenn Sie das Deep Learning-VM-Image bereitstellen, fügen Sie den kodierten Wert zur OVF-Eigenschaft `config-json` hinzu.

Fehlerbehebung bei der Bereitstellung von Deep Learning-VMs in VMware Private AI Foundation with NVIDIA

Die Fehlerbehebungsinformationen zur Bereitstellung einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA enthalten Lösungen für potenzielle Probleme, die auftreten können.

- Automatisierung von DL-Arbeitslasten wird nicht durchgeführt

  Nach dem Bereitstellen einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA wird die angegebene DL-Arbeitslast nicht ausgeführt.
- Fehler beim Herunterladen einer DL-Arbeitslast aufgrund ungültiger Anmeldedaten für die Authentifizierung

  Nach der Bereitstellung einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA schlägt das Herunterladen der angegebenen DL-Arbeitslast auf die virtuelle Maschine mit Fehlermeldungen fehl, die auf ungültige Anmeldedaten für die Authentifizierung hindeuten.
- Fehler beim Herunterladen des NVIDIA vGPU-Gasttreibers aufgrund eines fehlenden Download-Links

  Nach der Bereitstellung einer Deep Learning-VM schlägt das Herunterladen des angegebenen vGPU-Gasttreibers auf die virtuelle Maschine mit Fehlermeldungen fehl, die auf einen fehlenden Download-Link oder eine fehlende Ressource hindeuten.
- Der NVIDIA vGPU-Gasttreiber wird als „Nicht lizenziert“ angezeigt

  Nach der Bereitstellung einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA lautet der Status des NVIDIA vGPU-Gasttreibers auf „Nicht lizenziert“.

Automatisierung von DL-Arbeitslasten wird nicht durchgeführt

Nach dem Bereitstellen einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA wird die angegebene DL-Arbeitslast nicht ausgeführt.

Problem

Sie stellen eine Deep Learning-VM mit einer DL-Arbeitslast bereit, die beim ersten Start vorab installiert werden soll. Nach dem Start der Deep Learning-VM wird die DL-Arbeitslast nicht ausgeführt.

Ursache

1 Die base64-codierten `user-data` oder Werte anderer OVF-Parameter, wie z. B. `image-oneliner` oder `config-json`, werden in der Datei `/opt/dlvm/dl_app.sh` falsch gespeichert oder falsch dekodiert. Dies hat zur Folge, dass das DL-Arbeitslastskript nicht ausgeführt wird.

2 Die Installation des vGPU-Treibers ist fehlgeschlagen, wodurch das im OVF-Parameter `user-data` übergebene Cloud-init-Skript nicht ausgeführt wird. Das Cloud-init-Skript ist auf die erfolgreiche Installation des NVIDIA vGPU-Treibers angewiesen.

Lösung

Überprüfen Sie auf der Deep Learning-VM, ob die DL-Arbeitslast auf der virtuellen Maschine installiert ist, und wenden Sie eine passende Lösung an.

Verfügbarkeit der DL-Arbeitslast	Lösung
Die Komponenten der DL-Arbeitslast werden nicht auf der virtuellen Maschine erstellt.	■ Wenn Sie ein Cloud-init-Skript als Eingabe für den OVF-Parameter `user-data` verwenden, überprüfen Sie die folgenden Werte: ■ Überprüfen Sie das kodierte Skript, das als `user-data` eingegeben wird. Stellen Sie sicher, dass `#cloud-config` in der ersten Zeile angezeigt wird und im base64-Äquivalent enthalten ist. ■ Überprüfen Sie den Parameter `path`. ■ Überprüfen Sie die base64-codierte Zeichenfolge und stellen Sie sicher, dass der Wert `user-data` korrekt in `/opt/dlvm/dl_app.sh` gespeichert wurde. ■ Wenn Sie andere OVF-Parameter verwenden, überprüfen Sie die folgenden Werte: ■ `image-oneliner`. Überprüfen Sie die base64-codierte Zeichenfolge und stellen Sie sicher, dass der einzeilige Befehl korrekt in `/opt/dlvm/dl_app.sh` gespeichert wurde. ■ `config-json`. Überprüfen Sie die base64-codierte Zeichenfolge und stellen Sie sicher, dass die Docker-Erstellungsdatei und die Datei `config.json`, sofern angegeben, ordnungsgemäß in `/root/docker-compose.yaml` und `/root/.docker/config.json` gespeichert werden. Informationen zu den OVF-Parametern des aktuellen Deep Learning-VM-Images finden Sie unter OVF-Eigenschaften von Deep Learning-VMs.
Die Komponenten der DL-Arbeitslast werden erstellt, aber die Arbeitslast wird nicht ausgeführt.	■ Überprüfen Sie die Fehlermeldungen in `/var/log/vgpu-install.log`. ■ Stellen Sie bei Verwendung des Cloud-init-Skripts als Eingabe für den OVF-Parameter `user-data` sicher, dass der NVIDIA vGPU-Treiber installiert ist und ordnungsgemäß funktioniert. Das Cloud-init-Skript wird nicht ausgeführt, wenn die Installation des NVIDIA vGPU-Treibers fehlschlägt.

Fehler beim Herunterladen einer DL-Arbeitslast aufgrund ungültiger Anmeldedaten für die Authentifizierung

Nach der Bereitstellung einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA schlägt das Herunterladen der angegebenen DL-Arbeitslast auf die virtuelle Maschine mit Fehlermeldungen fehl, die auf ungültige Anmeldedaten für die Authentifizierung hindeuten.

Problem

Wenn Sie ein DL-Arbeitslast-Container-Image installieren, wie z. B. Triton Inference Server, TensorFlow oder Pytorch, enthält die Datei `/var/log/dl.log` folgende Meldung:

```
Unable to find image 'nvcr.io/nvidia/tritonserver-pb24h1:24.03.02-py3' locally docker: Error response from daemon: unauthorized: <html> <head><title>401 Authorization Required</title></head> <body>
```

Für NVIDIA RAG enthält die Datei `/var/log/dl.log` folgende Meldung:

```
Error: Invalid apikey chmod: cannot access 'llama2-13b-chat_vh100x2_fp16_24.02': No such file or directory Error: Invalid apikey chmod: cannot access 'nv-embed-qa_v4': No such file or directory stat /opt/data/rag-docker-compose_v24.03/docker-compose-vectordb.yaml: no such file or directory stat /opt/data/rag-docker-compose_v24.03/rag-app-text-chatbot.yaml: no such file or directory
```

Ursache

Die Authentifizierung bei der Containerregistrierung vom Typ „nvcr.io“ ist fehlgeschlagen. Folglich kann das DL-Arbeitslast-Image nicht auf die virtuelle Maschine heruntergeladen werden.

Lösung

- Überprüfen Sie die Anmeldedaten für die Anmeldung bei der als OVF-Parameter übergebenen Registrierung vom Typ „nvcr.io“ oder beim Assistenten für die Katalogeinrichtung für Private AI in VMware Aria Automation.
  - Registrierung: nvcr.io
  - Benutzerkonto für Registrierung: $oauthtoken
  - Registrierungskennwort: *API-Schlüssel des NGC-Portals*
- Stellen Sie sicher, dass der API-Schlüssel des NVIDIA NGC-Portals über die Berechtigungen zum Zugriff auf die erforderlichen Ressourcen verfügt und dass der Schlüssel nicht abgelaufen ist.

Fehler beim Herunterladen des NVIDIA vGPU-Gasttreibers aufgrund eines fehlenden Download-Links

Nach der Bereitstellung einer Deep Learning-VM schlägt das Herunterladen des angegebenen vGPU-Gasttreibers auf die virtuelle Maschine mit Fehlermeldungen fehl, die auf einen fehlenden Download-Link oder eine fehlende Ressource hindeuten.

Problem

Die Datei `/var/log/vgpu-install.log` enthält eine der folgenden Meldungen:

```
Fehler Kein Download-Link über API erkannt
```

```
Keine Downloads über API gefunden
```

Ursache

Der API-Schlüssel aus dem NVIDIA-Lizenzierungsportal, den Sie als Wert an die OVF-Eigenschaft `nvidia-portal-api-key` oder an den Assistenten für die Katalogeinrichtung für Private AI in VMware Aria Automation übergeben, ist ungültig, abgelaufen oder falsch formatiert.

Lösung

- Stellen Sie sicher, dass der API-Schlüssel gültig ist.
- Stellen Sie sicher, dass der API-Schlüssel korrekt eingegeben wurde.

Für API-Schlüssel wird in der Regel das Format der UUID-Version 4 `xxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx` verwendet.

Der NVIDIA vGPU-Gasttreiber wird als „Nicht lizenziert“ angezeigt

Nach der Bereitstellung einer Deep Learning-VM in VMware Private AI Foundation with NVIDIA lautet der Status des NVIDIA vGPU-Gasttreibers auf „Nicht lizenziert“.

Problem

Die Datei `/var/log/vgpu-install.log` enthält die folgenden Meldungen:

```
Lizenzstatus: Nicht lizenziert
```

```
Nicht lizenziert (eingeschränkt)
```

Ursache

Das NVIDIA vGPU-Clientkonfigurationstoken, das Sie als Wert an die OVF-Eigenschaft `vgpu-license` oder an den Assistenten für die Katalogeinrichtung für Private AI in VMware Aria Automation übergeben, ist ungültig, abgelaufen oder falsch formatiert.

Lösung

- Überprüfen Sie die Gültigkeit des Clientkonfigurationstokens.
- Stellen Sie sicher, dass die vGPU-Lizenz ordnungsgemäß formatiert ist und das JWT-Token-Format verwendet, das in der Regel folgendermaßen aussieht: `eyxxxx.eyxxxxx.xxxxx`.

  Sie können das JWT-Token unter jwt.io entschlüsseln, um das Ablaufdatum und die Knotenserver-URL zu überprüfen.
- Das vGPU-Lizenztoken wurde ebenfalls in `/etc/nvidia/ClientConfigToken/client_configuration_token.tok` gespeichert.
- Führen Sie zur weiteren Problembehebung diesen Befehl aus, um nach bestimmten Fehlermeldungen im Zusammenhang mit der Kommunikation des NVIDIA-Lizenzservers zu suchen.

  ```
  cat /var/log/syslog | grep -i nvidia
  ```

Führen Sie die folgenden Schritte aus, um ein neues Token anzuwenden:

1 Ersetzen Sie den Inhalt der Datei `/etc/nvidia/ClientConfigToken/client_configuration_token.tok` durch ein neues Token und führen Sie folgenden Befehl aus:

```
echo -n $vgpu_license_token > /etc/nvidia/ClientConfigToken/client_configuration_token.tok
```

2 Starten Sie den NVIDIA-Dienst neu.

```
/etc/init.d/nvidia-gridd restart
```

3 Überprüfen Sie den Lizenzstatus des NVIDIA vGPU-Gasttreibers.

```
nvidia-smi -q | grep -i "license status" | sed 's/^[ \t]*//'
```

Bereitstellen von KI-Arbeitslasten auf TKG-Clustern in VMware Private AI Foundation with NVIDIA

4

Als DevOps-Ingenieur können Sie KI-Containerarbeitslasten auf Tanzu Kubernetes Grid (TKG)-Clustern bereitstellen, deren Worker-Knoten mit NVIDIA GPUs beschleunigt werden.

Informationen zur Unterstützung von KI-Arbeitslasten auf TKG-Clustern finden Sie unter Informationen zum Bereitstellen von KI-/ML-Arbeitslasten auf TKGS-Clustern.

Lesen Sie als Nächstes die folgenden Themen:

- Bereitstellen eines GPU-beschleunigten TKG-Clusters mithilfe eines Self-Service-Katalogs in VMware Private AI Foundation with NVIDIA
- Bereitstellen eines GPU-beschleunigten TKG-Clusters mithilfe des `kubectl`-Befehls in einer verbundenen VMware Private AI Foundation with NVIDIA-Umgebung
- Bereitstellen eines GPU-beschleunigten TKG-Clusters mithilfe des `kubectl`-Befehls in einer getrennten VMware Private AI Foundation with NVIDIA-Umgebung

Bereitstellen eines GPU-beschleunigten TKG-Clusters mithilfe eines Self-Service-Katalogs in VMware Private AI Foundation with NVIDIA

In VMware Private AI Foundation with NVIDIA können Sie als DevOps-Ingenieur einen mit NVIDIA-GPUs beschleunigten TKG-Cluster aus VMware Aria Automation bereitstellen, indem Sie Self-Service-Katalogelemente eines KI-Kubernetes-Clusters in Automation Service Broker verwenden. Anschließend können Sie KI-Container-Images von NVIDIA NGC auf dem Cluster bereitstellen.

Voraussetzungen

Stellen Sie mit Ihrem Cloud-Administrator sicher, dass VMware Private AI Foundation with NVIDIA konfiguriert ist. Weitere Informationen finden Sie unter Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten.

Verfahren

- Stellen Sie in Automation Service Broker ein KI-Kubernetes-Cluster-Katalogelement auf der vom Cloud-Administrator konfigurierten Supervisor-Instanz bereit.
  - Verwenden Sie für einen Tanzu Grid Kubernetes-Cluster ohne RAG das Katalogelement **KI-Kubernetes-Cluster**. Weitere Informationen finden Sie unter Bereitstellen eines GPU-beschleunigten Tanzu Kubernetes Grid-Clusters.
  - Verwenden Sie für einen RAG-basierten Tanzu Grid Kubernetes-Cluster das Katalogelement **KI-Kubernetes-RAG-Cluster**. Weitere Informationen finden Sie unter Bereitstellen eines GPU-beschleunigten Tanzu Kubernetes Grid-RAG-Clusters.

Nächste Schritte

Führen Sie ein KI-Container-Image aus. Verwenden Sie in einer verbundenen Umgebung den NVIDIA NGC-Katalog. Verwenden Sie in einer getrennten Umgebung die Harbor-Registrierung auf dem Supervisor.

Stellen Sie für einen RAG-basierten Tanzu Grid Kubernetes-Cluster eine PostgreSQL-Datenbank vom Typ „pgvector“ in VMware Data Services Manager bereit und installieren Sie die RAG-Beispiel-Pipeline aus NVIDIA. Weitere Informationen finden Sie unter Bereitstellen einer RAG-Arbeitslast auf einem TKG-Cluster.

Bereitstellen eines GPU-beschleunigten TKG-Clusters mithilfe des `kubectl`-Befehls in einer verbundenen VMware Private AI Foundation with NVIDIA-Umgebung

In VMware Private AI Foundation with NVIDIA stellen Sie als DevOps-Ingenieur mithilfe der Kubernetes-API einen TKG-Cluster bereit, der NVIDIA-GPUs verwendet. Anschließend können Sie KI-Container-Arbeitslasten aus dem NVIDIA NGC-Katalog bereitstellen.

Sie verwenden `kubectl`, um den TKG-Cluster in dem vom Cloud-Administrator konfigurierten Namespace bereitzustellen.

Voraussetzungen

Stellen Sie mit dem Cloud-Administrator sicher, dass die folgenden Voraussetzungen für die KI-fähige Infrastruktur erfüllt sind.

- VMware Private AI Foundation with NVIDIA ist konfiguriert. Weitere Informationen finden Sie unter Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten.
- In einer getrennten Umgebung wird eine Inhaltsbibliothek mit Ubuntu-TKr-Images zum vSphere-Namespace für KI-Arbeitslasten hinzugefügt. Weitere Informationen finden Sie unter Konfigurieren einer Inhaltsbibliothek mit Ubuntu TKr für eine getrennte VMware Private AI Foundation with NVIDIA-Umgebung.

Verfahren

1 Melden Sie sich bei der Supervisor-Steuerungsebene an.

```
kubectl vsphere login --server=SUPERVISOR-CONTROL-PLANE-IP-ADDRESS-or-FQDN --vsphere-
username USERNAME
```

2 Stellen Sie einen TKG-Cluster bereit und installieren Sie den NVIDIA GPU-Operator und den NVIDIA-Netzwerkoperator darauf.

Weitere Informationen finden Sie unter Cluster-Operator-Workflow für die Bereitstellung von KI-/ML-Arbeitslasten auf TKGS-Clustern.

Nächste Schritte

Stellen Sie ein KI-Container-Image aus dem NVIDIA NGC-Katalog bereit.

Bereitstellen eines GPU-beschleunigten TKG-Clusters mithilfe des `kubectl`-Befehls in einer getrennten VMware Private AI Foundation with NVIDIA-Umgebung

In VMware Private AI Foundation with NVIDIA stellen Sie als DevOps-Ingenieur mithilfe der Kubernetes-API einen TKG-Cluster bereit, der NVIDIA-GPUs verwendet. In einer nicht verbundenen Umgebung müssen Sie zusätzlich ein lokales Ubuntu-Paket-Repository einrichten und die Harbor-Registrierung für den Supervisor verwenden.

Voraussetzungen

Stellen Sie mit dem Cloud-Administrator sicher, dass die folgenden Voraussetzungen für die KI-fähige Infrastruktur erfüllt sind.

- VMware Private AI Foundation with NVIDIA ist für eine getrennte Umgebung konfiguriert. Weitere Informationen finden Sie unter Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten.
- Eine Maschine mit Zugriff auf den Supervisor-Endpoint und auf das lokale Helm-Repository, auf denen die Diagrammdefinitionen des NVIDIA GPU-Operators gehostet werden.

Prozedur

1 Stellen Sie einen TKG-Cluster auf dem vom Cloud-Administrator konfigurierten vSphere-Namespace bereit.

Weitere Informationen finden Sie unter Bereitstellen eines TKGS-Clusters für NVIDIA vGPU.

2 Installieren Sie den NVIDIA GPU-Operator.

```
helm install --wait gpu-operator ./gpu-operator-4-1 -n gpu-operator
```

3 Überwachen Sie den Vorgang.

```
watch kubectl get pods -n gpu-operator
```

Nächste Schritte

Stellen Sie dem Supervisor ein KI-Container-Image aus der Harbor-Registrierung bereit.

Bereitstellen von RAG-Arbeitslasten in VMware Private AI Foundation with NVIDIA

5

Eine RAG-Arbeitslast (Retrieval-Augmented Generation) besteht aus einem LLM und einer externen Knowledgebase mit den neuesten Daten, die in einer Vektordatenbank gespeichert sind. In VMware Private AI Foundation with NVIDIA können Sie eine RAG-Arbeitslast für die Verwendung von Einbettungen aus einer Vektordatenbank konfigurieren, die von VMware Data Services Manager verwaltet wird.

Lesen Sie als Nächstes die folgenden Themen:

- Bereitstellen einer Vektordatenbank in VMware Private AI Foundation with NVIDIA
- Bereitstellen einer Deep Learning-VM mit einer RAG-Arbeitslast
- Bereitstellen einer RAG-Arbeitslast auf einem TKG-Cluster

Bereitstellen einer Vektordatenbank in VMware Private AI Foundation with NVIDIA

Wenn Sie die Retrieval-Augmented Generation (RAG) mit VMware Private AI Foundation with NVIDIA verwenden möchten, richten Sie mithilfe von VMware Data Services Manager eine PostgreSQL-Datenbank mit pgvector ein.

Sie können die Datenbank manuell erstellen oder einen Self-Service-Katalog in VMware Aria Automation erstellen, der von DevOps-Ingenieuren und Entwicklern verwendet werden kann.

Voraussetzungen

- Stellen Sie sicher, dass VMware Private AI Foundation with NVIDIA für die VI-Arbeitslastdomäne verfügbar ist. Weitere Informationen finden Sie unter Bereitstellen von VMware Private AI Foundation with NVIDIA.
- Stellen Sie mit Ihrem Cloud-Administrator sicher, dass die Voraussetzungen für das Erstellen einer PostgreSQL-Datenbank erfüllt sind. Weitere Informationen finden Sie unter Erstellen von Datenbanken.
- Installieren Sie das Befehlszeilendienstprogramm `psql` von der PostgreSQL-Website.

Verfahren

1 Stellen Sie eine PostgreSQL-Datenbank in der VI-Arbeitslastdomäne bereit und rufen Sie die Verbindungszeichenfolge für die Datenbank ab.

Sie können Sie einen der folgenden Workflows verwenden. Als Datenwissenschaftler können Sie eine Datenbank direkt über VMware Aria Automation bereitstellen. Andernfalls fordern Sie eine Datenbankbereitstellung bei Ihrem DSM-Administrator oder DSM-Benutzer an.

Bereitstellungsworkflow	Benötigte Benutzerrolle	Beschreibung
Stellen Sie die Verbindungszeichenfolge einer PostgreSQL-Datenbank bereit und rufen Sie sie aus VMware Aria Automation ab	Datenwissenschaftler oder DevOps-Ingenieur	Weitere Informationen finden Sie unter Bereitstellen einer Vektordatenbank mithilfe eines Self-Service-Katalogelements in VMware Aria Automation.
Stellen Sie die Verbindungszeichenfolge einer PostgreSQL-Datenbank über die VMware Data Services Manager-Konsole bereit und rufen Sie sie ab.	DSM-Administrator oder DSM-Benutzer oder ein Cloud-Administrator, dem eine dieser Rollen zugewiesen wurde	Weitere Informationen finden Sie unter Erstellen von Datenbanken und Herstellen einer Verbindung zu einer Datenbank.
Stellen Sie die Verbindungszeichenfolge einer PostgreSQL-Datenbank mithilfe des Befehls `kubectl` bereit und rufen Sie sie ab	DSM-Administrator oder DSM-Benutzer oder ein DevOps-Ingenieur, dem eine dieser Rollen zugewiesen wurde	Weitere Informationen finden Sie unter Aktivieren der Self-Service-Nutzung von VMware Data Services Manager.

Die Verbindungszeichenfolge der bereitgestellten Datenbank weist das folgende Format auf.

```
postgres://
pgvector_db_admin:encoded_pgvector_db_admin_password@pgvector_db_ip_address:5432/
pgvector_db_name
```

2 Aktivieren Sie die pgvector-Erweiterung in der Datenbank mithilfe des Befehlszeilendienstprogramms `psql`.

a Stellen Sie eine Verbindung zur Datenbank her.

```
psql -h pgvector_db_ip_address -p 5432 -d pgvector_db_name -U pgvector_db_admin -W
```

b Aktivieren Sie die pgvector-Erweiterung.

```
pgvector_db_name=# CREATE EXTENSION vector;
```

Nächste Schritte

Integrieren Sie die Datenbank in Ihre RAG-Arbeitslast. Weitere Informationen finden Sie unter Bereitstellen einer Deep Learning-VM mit einer RAG-Arbeitslast und Bereitstellen einer RAG-Arbeitslast auf einem TKG-Cluster.

Bereitstellen einer Vektordatenbank mithilfe eines Self-Service-Katalogelements in VMware Aria Automation

In VMware Private AI Foundation with NVIDIA können Sie als Datenwissenschaftler oder DevOps-Ingenieur eine Vektordatenbank aus VMware Aria Automation bereitstellen, indem Sie ein Self-Service-Katalogelement in Automation Service Broker verwenden.

Verfahren

1. Melden Sie sich bei VMware Aria Automation an und suchen Sie in Automation Service Broker das Katalogelement für die Datenbankbereitstellung gemäß den Informationen Ihres Cloud-Administrators.

   Standardmäßig wird das Katalogelement als **DSM DBaaS** bezeichnet.

2. Klicken Sie auf der Katalogelementkarte auf **Anforderung** und geben Sie die Details für die neue PostgreSQL-Datenbank ein.

   Weitere Informationen zu den Einstellungen für die Datenbank finden Sie unter Erstellen von Datenbanken.

3. Rufen Sie die Verbindungszeichenfolge der bereitgestellten Datenbank ab.

   a. Klicken Sie in Automation Service Broker auf **Bereitstellungen > Bereitstellungen**.

   b. Wählen Sie den Bereitstellungseintrag für die Datenbank aus.

   c. Wählen Sie auf der Registerkarte **Topologie** die Cloud-Vorlage für die Datenbankbereitstellung aus und wählen Sie im Menü **Aktionen** für die Vorlage die Option **Verbindungszeichenfolge abrufen** aus.

Ergebnisse

Weitere Informationen zur Bereitstellung und Durchführung von Vorgängen für Datenbanken in VMware Data Services Manager über VMware Aria Automation finden Sie in der Datei `readme.md` im Paket `AriaAutomation_DataServicesManager`.

Bereitstellen einer Deep Learning-VM mit einer RAG-Arbeitslast

Sie können eine Deep Learning-VM mit einer NVIDIA RAG-Arbeitslast unter Verwendung einer PostgreSQL-Datenbank vom Typ „pgvector“ bereitstellen, die von VMware Data Services Manager verwaltet wird.

Informationen zur NVIDIA RAG-Arbeitslast finden Sie in der Dokumentation zu NVIDIA RAG-Anwendungen mit Docker Compose (erfordert bestimmte Kontoberechtigungen).

Voraussetzungen

- Stellen Sie sicher, dass VMware Private AI Foundation with NVIDIA konfiguriert ist. Weitere Informationen finden Sie unter Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten.

- Bereitstellen einer Vektordatenbank in VMware Private AI Foundation with NVIDIA.

Verfahren

1 Wenn Sie als Datenwissenschaftler die Deep Learning-VM mithilfe eines Katalogelements in VMware Aria Automation bereitstellen, geben Sie die Details der PostgreSQL-Datenbank „pgvector“ an, nachdem Sie die virtuelle Maschine bereitgestellt haben.

   a Bereitstellen einer RAG-Workstation in VMware Aria Automation.

   b Navigieren Sie zu **Nutzung > Bereitstellungen > Bereitstellungen** und suchen Sie die Bereitstellung der Deep Learning-VM.

   c Speichern Sie im Abschnitt **Workstation-VM** die Details für die SSH-Anmeldung bei der virtuellen Maschine.

   d Melden Sie sich bei der Deep Learning-VM über SSH mit den in Automation Service Broker verfügbaren Anmeldedaten an.

   e Fügen Sie die folgenden pgvector-Variablen zur Datei `/opt/data/compose.env` hinzu:

```
POSTGRES_HOST_IP=pgvector_db_ip_address
POSTGRES_PORT_NUMBER=5432
POSTGRES_DB=pgvector_db_name
POSTGRES_USER=pgvector_db_admin
POSTGRES_PASSWORD=encoded_pgvector_db_admin_password
```

   f Starten Sie die NVIDIA RAG-Multi-Container-Anwendung neu, indem Sie die folgenden Befehle ausführen.

   Beispielsweise für NVIDIA RAG 24.03:

```
cd /opt/data
```

```
docker compose -f rag-docker-compose_v24.03/rag-app-text-chatbot.yaml down
```

```
docker compose -f rag-docker-compose_v24.03/docker-compose-vectordb.yaml down
```

```
docker compose -f rag-docker-compose_v24.03/docker-compose-vectordb.yaml up -d
```

2 Wenn Sie als DevOps-Ingenieur die Deep Learning-VM für einen Datenwissenschaftler direkt auf dem vSphere-Cluster oder mithilfe des Befehls `kubectl` bereitstellen, erstellen Sie ein cloud-init-Skript und stellen Sie die Deep Learning-VM bereit.

   a Erstellen Sie ein cloud-init-Skript für NVIDIA RAG und die von Ihnen erstellte pgvector PostgreSQL-Datenbank.

   Sie können die anfängliche Version des cloud-init-Skripts für NVIDIA RAG ändern. Beispielsweise für NVIDIA RAG 24.03 und eine pgvector-PostgreSQL-Datenbank mit Verbindungsdetails `postgres://` *`pgvector_db_admin`*`:`*`encoded_pgvector_db_admin_password`*`@`*`pgvector_db_ip_address`*`:5432/`*`pgvector_db_name`*.

```
#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    set -eu
    source /opt/dlvm/utils.sh
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    set_proxy "http" "https"

    cat <<EOF > /opt/dlvm/config.json
    {
      "_comment": "This provides default support for RAG: TensorRT inference,
llama2-13b model, and H100x2 GPU",
      "rag": {
        "org_name": "cocfwga8jq2c",
        "org_team_name": "no-team",
        "rag_repo_name": "nvidia/paif",
        "llm_repo_name": "nvidia/nim",
        "embed_repo_name": "nvidia/nemo-retriever",
        "rag_name": "rag-docker-compose",
        "rag_version": "24.03",
        "embed_name": "nv-embed-qa",
        "embed_type": "NV-Embed-QA",
        "embed_version": "4",
        "inference_type": "trt",
        "llm_name": "llama2-13b-chat",
        "llm_version": "h100x2_fp16_24.02",
        "num_gpu": "2",
        "hf_token": "huggingface token to pull llm model, update when using vllm
inference",
        "hf_repo": "huggingface llm model repository, update when using vllm inference"
      }
    }
    EOF
    CONFIG_JSON=$(cat "/opt/dlvm/config.json")
    INFERENCE_TYPE=$(echo "${CONFIG_JSON}" | jq -r '.rag.inference_type')
    if [ "${INFERENCE_TYPE}" = "trt" ]; then
      required_vars=("ORG_NAME" "ORG_TEAM_NAME" "RAG_REPO_NAME" "LLM_REPO_NAME"
```

```
"EMBED_REPO_NAME" "RAG_NAME" "RAG_VERSION" "EMBED_NAME" "EMBED_TYPE" "EMBED_VERSION"
"LLM_NAME" "LLM_VERSION" "NUM_GPU")
    elif [ "${INFERENCE_TYPE}" = "vllm" ]; then
      required_vars=("ORG_NAME" "ORG_TEAM_NAME" "RAG_REPO_NAME" "LLM_REPO_NAME"
"EMBED_REPO_NAME" "RAG_NAME" "RAG_VERSION" "EMBED_NAME" "EMBED_TYPE" "EMBED_VERSION"
"LLM_NAME" "NUM_GPU" "HF_TOKEN" "HF_REPO")
    else
      error_exit "Inference type '${INFERENCE_TYPE}' is not recognized. No action will
be taken."
    fi
    for index in "${!required_vars[@]}"; do
      key="${required_vars[$index]}"
      jq_query=".rag.${key,,} | select (.!=null)"
      value=$(echo "${CONFIG_JSON}" | jq -r "${jq_query}")
      if [[ -z "${value}" ]]; then
        error_exit "${key} is required but not set."
      else
        eval ${key}=\""${value}"\"
      fi
    done

    RAG_URI="${RAG_REPO_NAME}/${RAG_NAME}:${RAG_VERSION}"
    EMBED_MODEL_URI="${EMBED_REPO_NAME}/${EMBED_NAME}:${EMBED_VERSION}"

    NGC_CLI_VERSION="3.41.2"
    NGC_CLI_URL="https://api.ngc.nvidia.com/v2/resources/nvidia/ngc-apps/ngc_cli/
versions/${NGC_CLI_VERSION}/files/ngccli_linux.zip"

    mkdir -p /opt/data
    cd /opt/data

    if [ ! -f .file_downloaded ]; then
      # clean up
      rm -rf compose.env ${RAG_NAME}* ${LLM_NAME}* ngc* ${EMBED_NAME}*
*.json .file_downloaded

      # install ngc-cli
      wget --content-disposition ${NGC_CLI_URL} -O ngccli_linux.zip && unzip
ngccli_linux.zip
      export PATH=`pwd`/ngc-cli:${PATH}

      APIKEY=""
      REG_URI="nvcr.io"

      if [[ "$(grep registry-uri /opt/dlvm/ovf-env.xml | sed -n 's/.*oe:value="\
([^"]*\).*/\1/p')" == *"${REG_URI}"* ]]; then
        APIKEY=$(grep registry-passwd /opt/dlvm/ovf-env.xml | sed -n 's/.*oe:value="\
([^"]*\).*/\1/p')
      fi

      if [ -z "${APIKEY}" ]; then
          error_exit "No APIKEY found"
      fi

      # config ngc-cli
```

```
      mkdir -p ~/.ngc

      cat << EOF > ~/.ngc/config
      [CURRENT]
      apikey = ${APIKEY}
      format_type = ascii
      org = ${ORG_NAME}
      team = ${ORG_TEAM_NAME}
      ace = no-ace
    EOF

      # ngc docker login
      docker login nvcr.io -u \$oauthtoken -p ${APIKEY}

      # dockerhub login for general components, e.g. minio
      DOCKERHUB_URI=$(grep registry-2-uri /opt/dlvm/ovf-env.xml | sed -n
's/.*oe:value="\([^"]*\).*/\1/p')
      DOCKERHUB_USERNAME=$(grep registry-2-user /opt/dlvm/ovf-env.xml | sed -n
's/.*oe:value="\([^"]*\).*/\1/p')
      DOCKERHUB_PASSWORD=$(grep registry-2-passwd /opt/dlvm/ovf-env.xml | sed -n
's/.*oe:value="\([^"]*\).*/\1/p')

      if [[ -n "${DOCKERHUB_USERNAME}" && -n "${DOCKERHUB_PASSWORD}" ]]; then
        docker login -u ${DOCKERHUB_USERNAME} -p ${DOCKERHUB_PASSWORD}
      else
        echo "Warning: DockerHub not login"
      fi

      # get RAG files
      ngc registry resource download-version ${RAG_URI}

      # get llm model
      if [ "${INFERENCE_TYPE}" = "trt" ]; then
        LLM_MODEL_URI="${LLM_REPO_NAME}/${LLM_NAME}:${LLM_VERSION}"
        ngc registry model download-version ${LLM_MODEL_URI}
        chmod -R o+rX ${LLM_NAME}_v${LLM_VERSION}
        LLM_MODEL_FOLDER="/opt/data/${LLM_NAME}_v${LLM_VERSION}"
      elif [ "${INFERENCE_TYPE}" = "vllm" ]; then
        pip install huggingface_hub
        huggingface-cli login --token ${HF_TOKEN}
        huggingface-cli download --resume-download ${HF_REPO}/${LLM_NAME} --local-dir
${LLM_NAME} --local-dir-use-symlinks False
        LLM_MODEL_FOLDER="/opt/data/${LLM_NAME}"
        cat << EOF > ${LLM_MODEL_FOLDER}/model_config.yaml
        engine:
          model: /model-store
          enforce_eager: false
          max_context_len_to_capture: 8192
          max_num_seqs: 256
          dtype: float16
          tensor_parallel_size: ${NUM_GPU}
          gpu_memory_utilization: 0.8
    EOF
        chmod -R o+rX ${LLM_MODEL_FOLDER}
        python3 -c "import yaml, json, sys;
```

```
print(json.dumps(yaml.safe_load(sys.stdin.read())))" < "${RAG_NAME}_v${RAG_VERSION}/
rag-app-text-chatbot.yaml"> rag-app-text-chatbot.json
        jq '.services."nemollm-inference".image = "nvcr.io/nvidia/nim/nim_llm:24.02-
day0" |
            .services."nemollm-inference".command = "nim_vllm --model_name $
{MODEL_NAME} --model_config /model-store/model_config.yaml" |
            .services."nemollm-inference".ports += ["8000:8000"] |
            .services."nemollm-inference".expose += ["8000"]' rag-app-text-
chatbot.json > temp.json && mv temp.json rag-app-text-chatbot.json
        python3 -c "import yaml, json, sys; print(yaml.safe_dump(json.load(sys.stdin),
default_flow_style=False, sort_keys=False))" < rag-app-text-chatbot.json > "$
{RAG_NAME}_v${RAG_VERSION}/rag-app-text-chatbot.yaml"
      fi

      # get embedding models
      ngc registry model download-version ${EMBED_MODEL_URI}
      chmod -R o+rX ${EMBED_NAME}_v${EMBED_VERSION}

      # config compose.env
      cat << EOF > compose.env
      export MODEL_DIRECTORY="${LLM_MODEL_FOLDER}"
      export MODEL_NAME=${LLM_NAME}
      export NUM_GPU=${NUM_GPU}
      export APP_CONFIG_FILE=/dev/null
      export EMBEDDING_MODEL_DIRECTORY="/opt/data/${EMBED_NAME}_v${EMBED_VERSION}"
      export EMBEDDING_MODEL_NAME=${EMBED_TYPE}
      export EMBEDDING_MODEL_CKPT_NAME="${EMBED_TYPE}-${EMBED_VERSION}.nemo"
      export POSTGRES_HOST_IP=pgvector_db_ip_address
      export POSTGRES_PORT_NUMBER=5432
      export POSTGRES_DB=pgvector_db_name
      export POSTGRES_USER=pgvector_db_admin
      export POSTGRES_PASSWORD=encoded_pgvector_db_admin_password
    EOF

      touch .file_downloaded
    fi

    # start NGC RAG
    docker compose -f ${RAG_NAME}_v${RAG_VERSION}/docker-compose-vectordb.yaml up -d
pgvector
    source compose.env; docker compose -f ${RAG_NAME}_v${RAG_VERSION}/rag-app-text-
chatbot.yaml up -d

- path: /opt/dlvm/utils.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    error_exit() {
      echo "Error: $1" >&2
      vmtoolsd --cmd "info-set guestinfo.vmservice.bootstrap.condition false,
DLWorkloadFailure, $1"
      exit 1
    }

    check_protocol() {
```

```
      local proxy_url=$1
      shift
      local supported_protocols=("$@")
      if [[ -n "${proxy_url}" ]]; then
        local protocol=$(echo "${proxy_url}" | awk -F '://' '{if (NF > 1) print $1;
else print ""}')
        if [ -z "$protocol" ]; then
          echo "No specific protocol provided. Skipping protocol check."
          return 0
        fi
        local protocol_included=false
        for var in "${supported_protocols[@]}"; do
          if [[ "${protocol}" == "${var}" ]]; then
            protocol_included=true
            break
          fi
        done
        if [[ "${protocol_included}" == false ]]; then
          error_exit "Unsupported protocol: ${protocol}. Supported protocols are: $
{supported_protocols[*]}"
        fi
      fi
    }

    # $@: list of supported protocols
    set_proxy() {
      local supported_protocols=("$@")

      CONFIG_JSON_BASE64=$(grep 'config-json' /opt/dlvm/ovf-env.xml | sed -n
's/.*oe:value="\([^"]*\).*/\1/p')
      CONFIG_JSON=$(echo ${CONFIG_JSON_BASE64} | base64 --decode)

      HTTP_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r '.http_proxy // empty')
      HTTPS_PROXY_URL=$(echo "${CONFIG_JSON}" | jq -r '.https_proxy // empty')
      if [[ $? -ne 0 || (-z "${HTTP_PROXY_URL}" && -z "${HTTPS_PROXY_URL}") ]]; then
        echo "Info: The config-json was parsed, but no proxy settings were found."
        return 0
      fi

      check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}"
      check_protocol "${HTTPS_PROXY_URL}" "${supported_protocols[@]}"

      if ! grep -q 'http_proxy' /etc/environment; then
        echo "export http_proxy=${HTTP_PROXY_URL}
        export https_proxy=${HTTPS_PROXY_URL}
        export HTTP_PROXY=${HTTP_PROXY_URL}
        export HTTPS_PROXY=${HTTPS_PROXY_URL}
        export no_proxy=localhost,127.0.0.1" >> /etc/environment
        source /etc/environment
      fi

      # Configure Docker to use a proxy
      mkdir -p /etc/systemd/system/docker.service.d
      echo "[Service]
      Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\"
```

```
      Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\"
      Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/system/
docker.service.d/proxy.conf
      systemctl daemon-reload
      systemctl restart docker

      echo "Info: docker and system environment are now configured to use the proxy
settings"
    }
```

b Codieren Sie das cloud-init-Skript in das base64-Format.

Sie verwenden ein base64-Codierungstool, z. B. https://decode64base.com/, um die codierte Version Ihres cloud-init-Skripts zu generieren.

c Stellen Sie die Deep Learning-VM bereit und übergeben Sie den base64-Wert des cloud-init-Skripts an den Eingabeparameter `user-data`.

Siehe Direktes Bereitstellen einer Deep Learning-VM auf einem vSphere-Cluster in VMware Private AI Foundation with NVIDIA oder Bereitstellen einer Deep Learning-VM mithilfe des Befehls „kubectl“ in VMware Private AI Foundation with NVIDIA.

Bereitstellen einer RAG-Arbeitslast auf einem TKG-Cluster

Als DevOps-Ingenieur können Sie auf einem TKG-Cluster in einem Supervisor eine RAG-Arbeitslast auf Basis der RAG-Beispiel-Pipeline von NVIDIA bereitstellen, die eine von VMware Data Services Manager verwaltete PostgreSQL-Datenbank vom Typ „pgvector“ verwendet.

Voraussetzungen

- Stellen Sie sicher, dass VMware Private AI Foundation with NVIDIA für die VI-Arbeitslastdomäne verfügbar ist. Weitere Informationen finden Sie unter Kapitel 2 Vorbereiten von VMware Cloud Foundation für die Bereitstellung von Private AI-Arbeitslasten.
- Bereitstellen einer Vektordatenbank in VMware Private AI Foundation with NVIDIA.

Verfahren

1 Stellen Sie einen GPU-beschleunigten TKG-Cluster bereit.

Sie können Sie einen der folgenden Workflows verwenden.

Bereitstellungsworkflow	Schritte
Mithilfe eines Katalogelements in VMware Aria Automation	Bereitstellen eines GPU-beschleunigten Tanzu Kubernetes Grid-RAG-Clusters.
Mithilfe des Befehls `kubectl`	1 Stellen Sie einen GPU-beschleunigten TKG-Cluster mithilfe des Befehls `kubectl` bereit. ■ Informationen zu einer verbundenen Umgebung finden Sie unter Bereitstellen eines GPU-beschleunigten TKG-Clusters mithilfe des kubectl-Befehls in einer verbundenen VMware Private AI Foundation with NVIDIA-Umgebung. ■ Informationen zu einer getrennten Umgebung finden Sie unter Bereitstellen eines GPU-beschleunigten TKG-Clusters mithilfe des kubectl-Befehls in einer getrennten VMware Private AI Foundation with NVIDIA-Umgebung. 2 Installieren Sie den RAG LLM-Operator. Weitere Informationen finden Sie unter Installieren des RAG LLM-Operators.

2 Wenn Sie den Befehl `kubectl` zum Bereitstellen des TKG-Clusters verwendet haben, installieren Sie den NVIDIA RAG LLM-Operator auf dem TKG-Cluster.

   Weitere Informationen finden Sie unter Installieren des RAG LLM-Operators.

   Während der Bereitstellung wird mithilfe des Katalogelements **KI-Kubernetes-RAG-Cluster** in VMware Aria Automation der NVIDIA RAG LLM-Operator automatisch auf dem TKG-Cluster installiert.

3 Laden Sie die Manifeste für die NVIDIA-Beispiel-RAG-Pipeline herunter.

   Weitere Informationen finden Sie unter Beispiel-RAG-Pipeline.

4 Konfigurieren Sie die Beispiel-RAG-Pipeline mit der PostgreSQL-Datenbank „pgvector“.

   a Bearbeiten Sie eine Beispiel-Pipeline-YAML-Datei.

      Weitere Informationen finden Sie unter Schritt 4 in Beispiel-RAG-Pipeline.

   b Konfigurieren Sie in der YAML-Datei die Beispiel-Pipeline mit der PostgreSQL-Datenbank „pgvector“, indem Sie die Verbindungszeichenfolge der Datenbank verwenden.

      Weitere Informationen finden Sie unter Vektordatenbank für RAG-Beispiel-Pipeline.

5 Um eine externe IP-Adresse für die Beispiel-Chat-Anwendung anzugeben, legen Sie in der YAML-Datei `frontend.service.type` auf `loadBalancer` fest.

6 Starten Sie die Beispiel-RAG-Pipeline.

   Weitere Informationen finden Sie unter Beispiel-RAG-Pipeline.

7 Für den Zugriff auf die Beispiel-Chat-Anwendung führen Sie den folgenden Befehl aus, um die externe IP-Adresse der Anwendung abzurufen.

```
kubectl -n rag-sample get service rag-playground
```

8 Öffnen Sie in einem Webbrowser die Beispiel-Chat-Anwendung unter **http://*application_external_ip*:3001/orgs/nvidia/models/text-qa-chatbot**.

Überwachen von VMware Private AI Foundation with NVIDIA

6

Sie können GPU-Metriken auf Cluster- und Hostebene im vSphere Client und in VMware Aria Operations überwachen.

In VMware Aria Operations können Sie GPU-Metriken auf Cluster-, Hostsystem- und Hosteigenschaften-Ebene überwachen. Weitere Informationen finden Sie unter Private AI (GPU)-Dashboards und Eigenschaften für vCenter Server-Komponenten in VMware Aria Operations.

Im vSphere Client können Sie GPU-Metriken auf folgende Weise überwachen:

- Auf der Hostebene. Weitere Informationen finden Sie unter Leistungsdiagramme für Hosts in vSphere.
- Auf Clusterebene in benutzerdefinierten Diagrammen. Weitere Informationen finden Sie unter Arbeiten mit erweiterten und benutzerdefinierten Diagrammen in vSphere.