Guide de VMware Private AI Foundation with NVIDIA

23 juillet 2024
VMware Cloud Foundation 5.2

vmware
by Broadcom

Vous trouverez la documentation technique la plus récente sur le site Web de VMware by Broadcom, à l'adresse :

https://docs.vmware.com/fr/

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

Table des matières

À propos du Guide de VMware Private AI Foundation with NVIDIA

Le *Guide de VMware Private AI Foundation with NVIDIA* présente les composants de VMware Private AI Foundation with NVIDIA et les workflows généraux pour les cas d'utilisation de développement et de production.

Public visé

Les informations contenues dans le *Guide de VMware Private AI Foundation with NVIDIA* sont destinées aux administrateurs de cloud du centre de données, aux scientifiques des données et aux ingénieurs DevOps qui connaissent bien les éléments suivants :

- Administrateurs de cloud
  - Concepts de virtualisation et de centres de données définis par logiciel (SDDC)
  - Composants matériels, tels que les commutateurs Top of Rack (ToR), des commutateurs inter-rack, les serveurs avec un stockage en attachement direct, des câbles et des alimentations
  - Méthodes de configuration des GPU NVIDIA sur les serveurs dans un centre de données
  - Utilisation de VMware vSphere® à des fins de compatibilité avec des machines virtuelles.
  - Utilisation de vSphere IaaS control plane pour configurer et attribuer des ressources vSphere à des espaces de noms vSphere sur un superviseur.

  En tant qu'administrateur de cloud, reportez-vous aux informations suivantes :

  - Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI
  - Chapitre 3 Déploiement d'une VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA
  - Chapitre 6 Surveillance de VMware Private AI Foundation with NVIDIA
- Scientifiques des données
  - Conteneurs, y compris Docker, les graphiques Helm et le registre Harbor

  En tant que scientifique des données, reportez-vous aux informations suivantes :

  - Chapitre 3 Déploiement d'une VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA

    - Chapitre 5 Déploiement de charges de travail RAG dans VMware Private AI Foundation with NVIDIA
- Ingénieurs DevOps
    - Provisionnement de machines virtuelles dans vSphere à l'aide de l'API Kubernetes.
    - Conteneurs, y compris Docker, les graphiques Helm et le registre Harbor
    - Utilisation de vSphere IaaS control plane pour le provisionnement de VM et de clusters Tanzu Kubernetes Grid (TKG).

    En tant qu'ingénieur DevOps, reportez-vous aux informations suivantes :

    - Chapitre 4 Déploiement de charges de travail d'IA sur des clusters TKG dans VMware Private AI Foundation with NVIDIA
    - Chapitre 5 Déploiement de charges de travail RAG dans VMware Private AI Foundation with NVIDIA

Composants logiciels VMware

La fonctionnalité de la solution VMware Private AI Foundation with NVIDIA est disponible sur plusieurs composants logiciels en fonction de votre rôle dans l'organisation.

Rôle d'utilisateur cible	Catégorie de logiciel	Versions logicielles prises en charge
Administrateurs de cloud	Composants déployés dans VMware Cloud Foundation	Reportez-vous à la section Composants VMware dans VMware Private AI Foundation with NVIDIA.
Scientifiques des données	Composants de VM à apprentissage profond	Reportez-vous à la section Notes de mise à jour de VMware Deep Learning VM.
Ingénieurs DevOps	Versions de TK (TKr)	Reportez-vous à la section Notes de mise à jour de VMware Tanzu Kubernetes.

Documentation de VMware connexe

La solution VMware Private AI Foundation with NVIDIA inclut une pile de produits et de composants logiciels VMware. La documentation de ces produits logiciels est la suivante :

- Documentation de VMware Cloud Foundation
- Documentation de VMware vSphere et de vSAN
- Documentation de VMware vSphere IaaS Control Plane
- Documentation de VMware Aria Automation
- Documentation de VMware Aria Operations
- Documentation de VMware Aria Suite Lifecycle
- Documentation de VMware Data Services Manager

Glossaire de VMware Cloud Foundation

Le Glossaire de VMware Cloud Foundation définit les termes propres à VMware Cloud Foundation.

Qu'est-ce que VMware Private AI Foundation with NVIDIA ?

1

En tant que solution à composants multiples, vous pouvez utiliser VMware Private AI Foundation with NVIDIA pour exécuter des charges de travail d'IA génératives en utilisant l'informatique accélérée de NVIDIA, ainsi que la gestion de l'infrastructure virtuelle et la gestion du cloud à partir de VMware Cloud Foundation.

VMware Private AI Foundation with NVIDIA fournit une plate-forme pour le provisionnement de charges de travail d'IA sur des hôtes ESXi avec des GPU NVIDIA. En outre, l'exécution de charges de travail d'IA basées sur des conteneurs NVIDIA GPU Cloud (NGC) est spécifiquement validée par VMware.

VMware Private AI Foundation with NVIDIA prend en charge deux cas d'utilisation :

Cas d'utilisation de développement

Les administrateurs de cloud et les ingénieurs DevOps peuvent provisionner des charges de travail d'IA, y compris la génération augmentée de récupération (RAG), sous la forme de machines virtuelles à apprentissage profond. Les scientifiques des données peuvent utiliser ces machines virtuelles à apprentissage profond pour le développement de l'IA. Reportez-vous à la section À propos des images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA.

Cas d'utilisation de production

Les administrateurs de cloud peuvent fournir aux ingénieurs DevOps un environnement VMware Private AI Foundation with NVIDIA pour le provisionnement de charges de travail d'IA prêtes pour la production sur des clusters Tanzu Kubernetes Grid (TKG) dans vSphere IaaS control plane.

Pour plus d'informations sur les composants qui font partie de la solution VMware Private AI Foundation with NVIDIA et leur architecture au-dessus de VMware Cloud Foundation, reportez-vous à la section Architecture système de VMware Private AI Foundation with NVIDIA.

Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI

2

En tant qu'administrateur de cloud, vous devez déployer des logiciels spécifiques et configurer les domaines de charge de travail VI cibles afin que les scientifiques des données et les ingénieurs DevOps puissent déployer des charges de travail d'IA au-dessus de VMware Private AI Foundation with NVIDIA.

Composants VMware dans VMware Private AI Foundation with NVIDIA

La fonctionnalité de la solution VMware Private AI Foundation with NVIDIA est disponible sur plusieurs composants logiciels.

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

Pour plus d'informations sur l'architecture et les composants de VMware Private AI Foundation with NVIDIA, reportez-vous à la section Architecture système de VMware Private AI Foundation with NVIDIA.

Workflow de déploiement de VMware Private AI Foundation with NVIDIA

La fonctionnalité de VMware Private AI Foundation with NVIDIA est basée sur un ensemble de composants de base avec des composants supplémentaires requis pour activer le déploiement de l'un des types de charge de travail IA suivants :

- VM à apprentissage en profond en général
- Charges de travail d'IA sur un cluster TKG accéléré par GPU en général
- Charges de travail RAG en tant que VM ou applications à apprentissage profond sur des clusters TKG accélérés par GPU

Le déploiement d'une charge de travail RAG étend l'approche générale des VM à apprentissage profond et des charges de travail d'IA sur des clusters TKG avec le déploiement d'une base de données PostgreSQL pgvector et la configuration de l'application avec la base de données pgvector.

Dans un environnement déconnecté, vous devez prendre des mesures supplémentaires pour configurer et déployer des dispositifs, et fournir des ressources localement, afin que vos charges de travail puissent y accéder.

Environnement connecté

Tâche	Cas d'utilisation de déploiement de charges de travail d'IA	Étapes
Examinez l'architecture et les conditions requises pour le déploiement de VMware Private AI Foundation with NVIDIA.	Tous	■ Architecture système de VMware Private AI Foundation with NVIDIA ■ Conditions requises de déploiement de VMware Private AI Foundation with NVIDIA
Configurez une instance de service de licence sur le portail de licences NVIDIA et générez un jeton de configuration client.		Guide de l'utilisateur du système de licence NVIDIA.
Générez une clé API pour accéder au catalogue NVIDIA NGC.		Extraction et exécution de conteneurs NVIDIA AI Enterprise
Créez une bibliothèque de contenu pour les images de VM à apprentissage profond.	Déployer une VM à apprentissage profond	Créer une bibliothèque de contenu avec des images de VM à apprentissage profond pour VMware Private AI Foundation with NVIDIA
Activez vSphere IaaS control plane (anciennement vSphere with Tanzu).	Tous	Configurer vSphere IaaS Control Plane pour VMware Private AI Foundation with NVIDIA
Déployer Déployer VMware Aria Automation à l'aide de VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.	Tous Requis si les scientifiques des données et les ingénieurs DevOps déploient des charges de travail à l'aide d'éléments de catalogue en libre-service dans VMware Aria Automation.	1 Automatisation du cloud privé pour VMware Cloud Foundation 2 Configurer VMware Aria Automation pour VMware Private AI Foundation with NVIDIA
Déployez VMware Aria Operations à l'aide de VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.	Tous	Gestion intelligente des opérations pour VMware Cloud Foundation.

Tâche	Cas d'utilisation de déploiement de charges de travail d'IA	Étapes
Déployer VMware Data Services Manager	Déployer une charge de travail RAG	1 Installation et configuration de VMware Data Services Manager Déployez une instance de VMware Data Services Manager dans le domaine de gestion. 2 Créer un élément de catalogue de base de données vectorielle dans VMware Aria Automation
Configurez une machine qui a accès à l'instance de superviseur et qui dispose de Docker, Helm et Kubernetes CLI Tools for vSphere.	Tous Requis si les charges de travail d'IA sont déployées directement à l'aide de la commande `kubectl`.	Installer Kubernetes CLI Tools for vSphere

Environnement déconnecté

Tâche	Options de déploiement des charges de travail d'IA associées	Étapes
Vérifiez la configuration requise pour le déploiement de VMware Private AI Foundation with NVIDIA.	Tous	■ Architecture système de VMware Private AI Foundation with NVIDIA ■ Conditions requises de déploiement de VMware Private AI Foundation with NVIDIA
Déployez une instance de NVIDIA Delegated License Service.		Installation et configuration du dispositif virtuel DLS Vous pouvez déployer le dispositif virtuel dans le même domaine de charge de travail que les charges de travail d'IA ou dans le domaine de gestion.
1 Enregistrez une instance de NVIDIA DLS sur le portail de licences NVIDIA, puis liez et installez un dispositif License Server sur celui-ci. 2 Générez un jeton de configuration client.		■ Configuration d'une instance de service ■ Gestion des licences sur un dispositif License Server.
Créer une bibliothèque de contenu pour les images de VM à apprentissage profond	Déployer une VM à apprentissage profond	Créer une bibliothèque de contenu avec des images de VM à apprentissage profond pour VMware Private AI Foundation with NVIDIA
Activer vSphere IaaS control plane (anciennement vSphere with Tanzu)	Tous	Configurer vSphere IaaS Control Plane pour VMware Private AI Foundation with NVIDIA

Tâche	Options de déploiement des charges de travail d'IA associées	Étapes
■ Configurez une machine qui a accès à Internet et sur laquelle Docker et Helm sont installés. ■ Configurez une machine qui a accès à vCenter Server pour le domaine de charge de travail VI, l'instance de superviseur et le registre de conteneur local. La machine doit disposer de Docker, Helm et Kubernetes CLI Tools for vSphere.		■ Déploiement d'un hôte bastion ■ Installer Kubernetes CLI Tools for vSphere
Configurer une bibliothèque de contenu pour les versions de Tanzu Kubernetes (TKr) pour Ubuntu	■ Déployer une charge de travail RAG sur un cluster TKG accéléré par GPU ■ Déployer des charges de travail d'IA sur un cluster TKG accéléré par GPU	Configurer une bibliothèque de contenu avec Ubuntu TKr pour un environnement VMware Private AI Foundation with NVIDIA déconnecté
Configurez un service de registre Harbor dans le superviseur.	Tous Requis si les charges de travail d'IA sont déployées sur un superviseur dans vSphere IaaS control plane Dans un environnement sans vSphere IaaS control plane, pour extraire des images de conteneur sur une VM à apprentissage profond s'exécutant directement sur un cluster vSphere, vous devez configurer un registre à partir d'un autre fournisseur.	Configuration d'un registre Harbor privé dans VMware Private AI Foundation with NVIDIA
Chargez les composants des opérateurs NVIDIA dans l'environnement.	■ Déployer une charge de travail RAG sur un cluster TKG accéléré par GPU ■ Déployer des charges de travail d'IA sur un cluster TKG accéléré par GPU	Charger les composants de l'opérateur NVIDIA GPU dans un environnement déconnecté

Tâche	Options de déploiement des charges de travail d'IA associées	Étapes
Fournissez un emplacement à partir duquel télécharger les pilotes invités vGPU.	Déployer une VM à apprentissage profond	Effectuez le chargement vers un serveur Web local des versions des pilotes invités vGPU requises, téléchargées à partir du portail de licences NVIDIA et d'un index dans l'un des formats suivants : ■ Fichier d'index `.txt` avec la liste des fichiers `.run` ou `.zip` des pilotes invités vGPU. `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` ■ Index d'annuaire au format généré par les serveurs Web, tels que NGINX et Apache HTTP Server. Les fichiers de pilote vGPU spécifiques à la version doivent être fournis sous forme de fichiers `.zip`.
Chargez les images du conteneur NVIDIA NGC dans un registre de conteneur privé, tel que le service de registre Harbor du superviseur.	Tous Dans un environnement sans vSphere IaaS control plane, pour extraire des images de conteneur sur une VM à apprentissage profond s'exécutant directement sur un cluster vSphere, vous devez configurer un registre à partir d'un autre fournisseur.	Charger des images de conteneur AI dans un registre Harbor privé de VMware Private AI Foundation with NVIDIA
Déployez VMware Aria Automation à l'aide de VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.	Tous Requis si les scientifiques des données et les ingénieurs DevOps déploient des charges de travail à l'aide d'éléments de catalogue en libre-service dans VMware Aria Automation.	1 Automatisation du cloud privé pour VMware Cloud Foundation 2 Configurer VMware Aria Automation pour VMware Private AI Foundation with NVIDIA

Tâche	Options de déploiement des charges de travail d'IA associées	Étapes
Déployez VMware Aria Operations à l'aide de VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.	Tous	Gestion intelligente des opérations pour VMware Cloud Foundation
Déployer VMware Data Services Manager	Déployer une charge de travail RAG	1 Installation et configuration de VMware Data Services Manager Déployez une instance de VMware Data Services Manager dans le domaine de gestion. 2 Créer un élément de catalogue de base de données vectorielle dans VMware Aria Automation

Lisez les sections suivantes :

- Architecture système de VMware Private AI Foundation with NVIDIA
- Conditions requises de déploiement de VMware Private AI Foundation with NVIDIA
- Créer une bibliothèque de contenu avec des images de VM à apprentissage profond pour VMware Private AI Foundation with NVIDIA
- Configurer vSphere IaaS Control Plane pour VMware Private AI Foundation with NVIDIA
- Configurer une bibliothèque de contenu avec Ubuntu TKr pour un environnement VMware Private AI Foundation with NVIDIA déconnecté
- Configuration d'un registre Harbor privé dans VMware Private AI Foundation with NVIDIA
- Charger les composants de l'opérateur NVIDIA GPU dans un environnement déconnecté
- Configurer VMware Aria Automation pour VMware Private AI Foundation with NVIDIA

Architecture système de VMware Private AI Foundation with NVIDIA

VMware Private AI Foundation with NVIDIA s'exécute en plus de l'ajout de la prise en charge de VMware Cloud Foundation pour toutes les charges de travail d'IA dans les domaines de charge de travail VI avec vSphere IaaS control plane provisionné à l'aide de kubectl et de VMware Aria Automation.

Figure 2-1. Exemple d'architecture pour VMware Private AI Foundation with NVIDIA

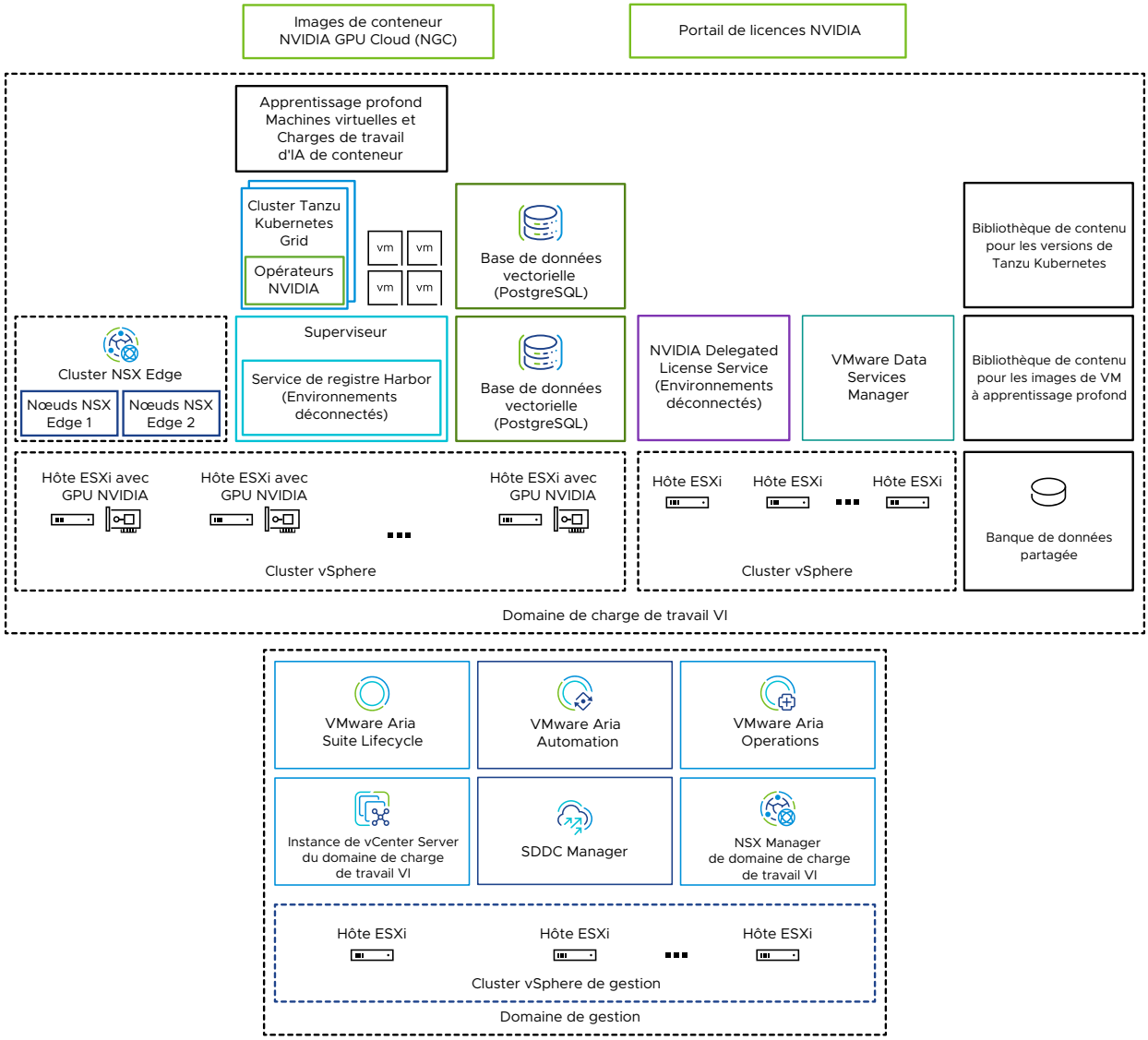


Tableau 2-1. Composants pour l'exécution de charges de travail d'IA dans VMware Private AI Foundation with NVIDIA

Composant	Description
Hôtes ESXi avec GPU activé	Hôtes ESXi configurés de la manière suivante : ■ Le GPU NVIDIA doit être pris en charge pour VMware Private AI Foundation with NVIDIA. Le GPU est partagé entre les charges de travail à l'aide du découpage temporel ou du mécanisme de GPU multi-instances (MIG). ■ Le pilote du gestionnaire d'hôte NVIDIA vGPU doit être installé afin que vous puissiez utiliser des profils vGPU basés sur MIG ou le découpage temporel.
Superviseur	Un ou plusieurs clusters vSphere doivent être activés pour vSphere IaaS control plane afin que vous puissiez exécuter des machines virtuelles et des conteneurs sur vSphere à l'aide de l'API Kubernetes. Un superviseur est un cluster Kubernetes, servant de plan de contrôle pour gérer les clusters de charge de travail et les machines virtuelles.
Registre Harbor	Registre d'images local dans un environnement déconnecté dans lequel vous hébergez les images de conteneur téléchargées à partir du catalogue NVIDIA NGC.
Cluster NSX Edge	Cluster de nœuds NSX Edge qui fournit un routage nord-sud à 2 niveaux pour le superviseur et les charges de travail qu'il exécute. La passerelle de niveau 0 sur le cluster NSX Edge est en mode actif-actif.
Opérateurs NVIDIA	■ Opérateur NVIDIA GPU. Automatise la gestion de tous les composants logiciels NVIDIA requis pour provisionner le GPU dans les conteneurs d'un cluster Kubernetes. L'opérateur NVIDIA GPU est déployé sur un cluster TKG. ■ Opérateur réseau NVIDIA. L'opérateur réseau NVIDIA permet également de configurer les pilotes mellanox appropriés pour les conteneurs à l'aide de fonctions virtuelles pour la mise en réseau haut débit, RDMA et GPUDirect. L'opérateur réseau fonctionne conjointement avec l'opérateur GPU pour activer GPUDirect RDMA sur des systèmes compatibles. L'opérateur réseau NVIDIA est déployé sur un cluster TKG.
Base de données vectorielle	Base de données PostgreSQL sur laquelle l'extension pgvector est activée afin que vous puissiez l'utiliser dans les charges de travail d'IA de génération augmentée de récupération (RAG).

Tableau 2-1. Composants pour l'exécution de charges de travail d'IA dans VMware Private AI Foundation with NVIDIA (suite)

Composant	Description
■ Portail de licences NVIDIA ■ NVIDIA Delegated License Service (DLS)	Utilisez le portail de licences NVIDIA pour générer un jeton de configuration client afin d'attribuer une licence au pilote vGPU invité dans la machine virtuelle à apprentissage profond et les opérateurs GPU sur les clusters TKG. Dans un environnement déconnecté ou pour que vos charges de travail obtiennent des informations de licence sans utiliser de connexion Internet, hébergez les licences NVIDIA localement sur un dispositif DLS (Delegated License Service).
Bibliothèque de contenu	Les bibliothèques de contenu stockent les images pour les machines virtuelles à apprentissage profond et pour les versions de Tanzu Kubernetes. Utilisez ces images pour le déploiement de charges de travail d'IA dans l'environnement VMware Private AI Foundation with NVIDIA. Dans un environnement connecté, les bibliothèques de contenu extraient leur contenu des bibliothèques de contenu publiques gérées de VMware. Dans un environnement déconnecté, vous devez charger manuellement les images requises ou les extraire d'un serveur miroir de bibliothèque de contenu interne.
Catalogue NVIDIA GPU Cloud (NGC)	Portail des conteneurs optimisés pour le GPU de l'IA et de l'apprentissage automatique qui sont testés et prêts à s'exécuter sur des GPU NVIDIA pris en charge sur site au-dessus de VMware Private AI Foundation with NVIDIA.

En tant qu'administrateur de cloud, utilisez les composants de gestion dans VMware Cloud Foundation

Tableau 2-2. Composants de gestion dans VMware Private AI Foundation with NVIDIA

Composant de gestion	Description
SDDC Manager	Utilisez SDDC Manager pour les tâches suivantes : ■ Déployez un domaine de charge de travail VI avec GPU activé basé sur des images vSphere Lifecycle Manager et ajoutez-y des clusters. ■ Déployez un cluster NSX Edge dans des domaines de charge de travail VI à utiliser par les instances de superviseur et dans le domaine de gestion pour les composants de VMware Aria Suite de VMware Private AI Foundation with NVIDIA. ■ Déployez une instance de VMware Aria Suite Lifecycle intégrée au référentiel SDDC Manager.
Instance de vCenter Server de domaine de charge de travail VI	Utilisez cette instance de vCenter Server pour activer et configurer un superviseur.
NSX Manager de domaine de charge de travail VI	SDDC Manager utilise cette instance de NSX Manager pour déployer et mettre à jour des clusters NSX Edge.
VMware Aria Suite Lifecycle	Utilisez VMware Aria Suite Lifecycle pour déployer et mettre à jour VMware Aria Automation, ainsi que VMware Aria Operations.
VMware Aria Automation	Utilisez VMware Aria Automation pour ajouter des éléments de catalogue en libre-service pour le déploiement de charges de travail d'IA pour les ingénieurs DevOps et les scientifiques des données.
VMware Aria Operations	Utilisez VMware Aria Operations pour surveiller la consommation de GPU dans les domaines de charge de travail avec GPU activé.
VMware Data Services Manager	Utilisez VMware Data Services Manager pour créer des bases de données vectorielles, telles qu'une base de données PostgreSQL avec l'extension pgvector.

Conditions requises de déploiement de VMware Private AI Foundation with NVIDIA

Déployez des composants de VMware Private AI Foundation with NVIDIA de votre environnement VMware Cloud Foundation dans un domaine de charge de travail VI sur lequel certains composants NVIDIA doivent être installés.

Versions logicielles VMware requises

Reportez-vous à la section Composants VMware dans VMware Private AI Foundation with NVIDIA.

Périphériques GPU NVIDIA pris en charge

Avant d'utiliser VMware Private AI Foundation with NVIDIA, assurez-vous que les GPU sur vos hôtes ESXi sont pris en charge par VMware by Broadcom :

Tableau 2-3. Composants NVIDIA pris en charge pour VMware Private AI Foundation with NVIDIA

Composant NVIDIA	Options prises en charge
GPU NVIDIA	■ NVIDIA A100 ■ NVIDIA L40S ■ NVIDIA H100
Mode de partage de GPU	■ Découpage temporel ■ GPU multi-instances (MIG)

Logiciel NVIDIA requis

Le périphérique GPU doit prendre en charge les derniers profils vGPU de NVIDIA AI Enterprise (NVAIE). Pour plus d'informations, reportez-vous à la documentation des GPU pris en charge par le logiciel NVIDIA Virtual GPU.

- Pilote d'hôte NVIDIA vGPU (y compris le VIB pour les hôtes ESXi), compatible avec votre version de VMware Cloud Foundation. Reportez-vous aux Notes de mise à jour du logiciel Virtual GPU pour VMware vSphere.
- Opérateur NVIDIA GPU compatible avec la version Kubernetes des clusters TKG déployés. Reportez-vous aux Notes de mise à jour de l'opérateur NVIDIA GPU et à Notes de mise à jour de VMware Tanzu Kubernetes.

Configuration requise de VMware Cloud Foundation

Avant de déployer VMware Private AI Foundation with NVIDIA, une configuration spécifique doit être disponible dans VMware Cloud Foundation.

- Une licence VMware Cloud Foundation.
- Une licence de module complémentaire VMware Private AI Foundation with NVIDIA.

  Vous avez besoin de la licence de module complémentaire VMware Private AI Foundation with NVIDIA pour accéder aux fonctionnalités suivantes :

  - La configuration de Private AI dans VMware Aria Automation pour les éléments de catalogue afin de faciliter le provisionnement de machines virtuelles à apprentissage profond accélérées par GPU et de clusters TKG.
  - Provisionnement de bases de données PostgreSQL à l'aide de l'extension pgvector avec l'assistance de l'entreprise.
  - Déploiement et utilisation de l'image de machine virtuelle à apprentissage profond fournie par VMware by Broadcom.

Vous pouvez déployer des charges de travail d'IA avec et sans superviseur activé, et utiliser les mesures de GPU dans vCenter Server et VMware Aria Operations avec la licence VMware Cloud Foundation.

- Produit NVIDIA vGPU sous licence incluant le fichier VIB du pilote d'hôte pour les hôtes ESXi et les pilotes du SE invité. Pour plus d'informations, reportez-vous à la documentation des GPU pris en charge par le logiciel NVIDIA Virtual GPU.
- Fichier VIB du pilote d'hôte NVIDIA vGPU téléchargé à partir de https://nvid.nvidia.com/
- Une image vSphere Lifecycle Manager avec le fichier VIB du pilote du gestionnaire d'hôte vGPU disponible dans SDDC Manager. Reportez-vous à la section Gestion des images de vSphere Lifecycle Manager dans VMware Cloud Foundation.
- Un domaine de charge de travail VI avec au moins 3 hôtes ESXi avec GPU activé qui est basé sur l'image vSphere Lifecycle Manager contenant le fichier VIB du pilote du gestionnaire d'hôte. Reportez-vous aux sections Déployer un domaine de charge de travail VI à l'aide de l'interface utilisateur de SDDC Manager et Gestion des images de vSphere Lifecycle Manager dans VMware Cloud Foundation.
- Pilote d'hôte NVIDIA vGPU installé et vGPU configuré sur chaque hôte ESXi dans le cluster pour les charges de travail d'IA.

  a Sur chaque hôte ESXi, activez SR-IOV dans le BIOS et Partagés en direct sur les périphériques graphiques pour les opérations d'IA.

  Pour plus d'informations sur la configuration de SR-IOV, reportez-vous à la documentation de votre fournisseur du matériel. Pour plus d'informations sur la configuration de Partagés en direct sur les périphériques graphiques, reportez-vous à la section Configurer des graphiques virtuels sur vSphere.

  b Installez le pilote du gestionnaire d'hôte NVIDIA vGPU sur chaque hôte ESXi de l'une des manières suivantes :

  - Installez le pilote sur chaque hôte et ajoutez le fichier VIB du pilote à l'image vSphere Lifecycle du cluster.

    Reportez-vous au Guide de démarrage rapide du logiciel NVIDIA Virtual GPU.

  - Ajoutez le fichier VIB du pilote à l'image vSphere Lifecycle du cluster et corrigez les hôtes.

  c Pour utiliser le partage de GPU multi-instances (MIG), activez-le sur chaque hôte ESXi du cluster.

  Reportez-vous à la section Guide de l'utilisateur de NVIDIA MIG.

  d Sur l'instance de vCenter Server du domaine de charge de travail VI, définissez le paramètre avancé `vgpu.hotmigrate.enabled` sur `true` afin de migrer les machines virtuelles avec vGPU à l'aide de vSphere vMotion.

  Reportez-vous à la section Configurer les paramètres avancés.

Créer une bibliothèque de contenu avec des images de VM à apprentissage profond pour VMware Private AI Foundation with NVIDIA

Les images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA sont distribuées dans une bibliothèque de contenu partagée publiée par VMware. En tant qu'administrateur de cloud, utilisez une bibliothèque de contenu pour extraire des images de VM spécifiques dans votre domaine de charge de travail VI lors du déploiement de VM.

Conditions préalables

En tant qu'administrateur de cloud, vérifiez que VMware Private AI Foundation with NVIDIA est déployé et configuré. Reportez-vous à la section Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI.

Procédure

1 Connectez-vous à l'instance de vCenter Server pour le domaine de charge de travail VI à l'adresse **`https://<vcenter_server_fqdn>/ui`**.

2 Sélectionnez **Menu > Bibliothèques de contenu** et cliquez sur **Créer**.

3 Créez une bibliothèque de contenu pour les images de VM à apprentissage profond.

- Pour un environnement connecté, créez une bibliothèque de contenu abonnée connectée à `https://packages.vmware.com/dl-vm/lib.json`. L'authentification n'est pas requise.
- Pour un environnement déconnecté, téléchargez les images de VM à apprentissage profond à partir de `https://packages.vmware.com/dl-vm/` et importez-les dans une bibliothèque de contenu locale.

  Pour chaque image, téléchargez les fichiers `.ovf`, `.vmdk`, `.mf` et `.cert` appropriés.

Configurer vSphere IaaS Control Plane pour VMware Private AI Foundation with NVIDIA

Pour permettre aux ingénieurs DevOps et aux scientifiques des données de déployer des machines virtuelles à apprentissage profond ou des clusters TKG avec des charges de travail de conteneur IA, vous devez déployer un superviseur sur un cluster avec GPU activé dans un domaine de charge de travail VI et créer des classes de VM avec vGPU activé.

Conditions préalables

Reportez-vous à Conditions requises de déploiement de VMware Private AI Foundation with NVIDIA.

Procédure

1 Déployez un cluster NSX Edge dans le domaine de charge de travail VI à l'aide de SDDC Manager.

   SDDC Manager déploie également une passerelle de niveau 0 que vous spécifiez lors du déploiement du superviseur. La passerelle de niveau 0 est en mode de haute disponibilité actif-actif.

2 Configurez une stratégie de stockage pour le superviseur.

   Reportez-vous à la section Créer des stratégies de stockage pour vSphere with Tanzu.

3 Déployez un superviseur sur un cluster d'hôtes ESXi avec GPU activé dans le domaine de charge de travail VI.

   Utilisez l'attribution d'adresses IP statiques pour le réseau de gestion. Attribuez le réseau de gestion de VM du superviseur sur le commutateur vSphere Distributed Switch pour le cluster.

   Configurez le réseau de charge de travail de la manière suivante :

   - Utilisez le commutateur vSphere Distributed Switch pour le cluster ou créez-en un spécifiquement pour les charges de travail d'IA.
   - Configurez le superviseur avec le cluster NSX Edge et la passerelle de niveau 0 que vous avez déployés à l'aide de SDDC Manager.
   - Définissez les valeurs restantes en fonction de votre conception.

   Utilisez la stratégie de stockage que vous avez créée.

   Pour plus d'informations sur le déploiement d'un superviseur sur un cluster unique, reportez-vous à la section Déployer un superviseur à une seule zone avec la mise en réseau NSX.

4 Configurez les classes de VM basées sur vGPU pour les charges de travail d'IA.

   Dans ces classes de VM, définissez le calcul requis et un profil vGPU pour un périphérique NVIDIA GRID vGPU en fonction des périphériques vGPU configurés sur les hôtes ESXi dans le cluster superviseur.

   - Pour plus d'informations sur la configuration de classes de VM basées sur vGPU pour les machines virtuelles, reportez-vous aux sections Créer une classe de VM personnalisée à l'aide de vSphere Client et Ajouter des périphériques PCI à une classe de VM dans vSphere with Tanzu.
   - Pour plus d'informations sur la configuration des classes de VM basées sur vGPU pour les nœuds worker TKG, reportez-vous aux sections Créer une classe de VM personnalisée avec un profil vGPU dans vSphere 8 Update 2b et versions ultérieures et Configuration d'espaces de noms vSphere pour les clusters TKG sur le superviseur.

Pour la classe de VM du déploiement de VM à apprentissage profond avec des charges de travail NVIDIA RAG, définissez les paramètres supplémentaires suivants dans la boîte de dialogue Classe de VM :

- Sélectionnez le profil vGPU complet pour le mode de découpage temporel ou un profil MIG. Par exemple, pour la carte NVIDIA A100 40 Go en mode de découpage temporel vGPU, sélectionnez **nvidia_a100-40c**.
- Dans l'onglet **Matériel virtuel**, allouez plus de 16 cœurs de CPU virtuels et 64 Go de mémoire virtuelle.
- Dans l'onglet **Paramètres avancés**, définissez le paramètre `pciPassthru<vgpu-id>.cfg.enable_uvm` sur **1**.

  où `<vgpu-id>` identifie le vGPU attribué à la machine virtuelle. Par exemple, si deux vGPU sont attribués à la machine virtuelle, définissez `pciPassthru0.cfg.parameter=1` et `pciPassthru1.cfg.parameter = 1`.

5 Si vous prévoyez d'utiliser l'outil de ligne de commande `kubectl` pour déployer une VM à apprentissage profond ou un cluster TKG accéléré par GPU sur un superviseur, créez et configurez un espace de noms vSphere, en ajoutant des limites de ressources, une stratégie de stockage, des autorisations pour les ingénieurs DevOps et en y associant les classes de VM basées sur vGPU.

- Pour plus d'informations sur la configuration des espaces de noms vSphere pour les machines virtuelles, reportez-vous à la section Créer et configurer un espace de noms vSphere sur le superviseur.
- Pour plus d'informations sur la configuration des espaces de noms vSphere pour les clusters TKG, reportez-vous à la section Configuration d'espaces de noms vSphere pour les clusters TKG sur le superviseur.

6 Si vous prévoyez d'activer les déploiements de VM à apprentissage profond sur un superviseur en appelant directement `kubectl`, ajoutez la bibliothèque de contenu à l'espace de noms vSphere pour les charges de travail d'IA.

VMware Aria Automation crée un espace de noms lors du provisionnement d'une VM à apprentissage profond, en y ajoutant automatiquement la bibliothèque de contenu.

a Sélectionnez **Menu > Gestion de la charge de travail**.

b Accédez à l'espace de noms pour les charges de travail d'IA.

c Dans la carte **Service de VM**, cliquez sur **Gérer les bibliothèques de contenu**.

d Sélectionnez la bibliothèque de contenu avec les images de VM à apprentissage profond, puis cliquez sur **OK**.

Configurer une bibliothèque de contenu avec Ubuntu TKr pour un environnement VMware Private AI Foundation with NVIDIA déconnecté

En tant qu'administrateur de cloud, si votre environnement ne dispose pas d'une connectivité Internet, fournissez une bibliothèque de contenu locale dans laquelle vous chargez manuellement les versions de Tanzu Kubernetes (TKr) et l'associez au superviseur.

Le déploiement de charges de travail d'IA prenant en charge NVIDIA sur des clusters TKG nécessite l'utilisation de l'édition Ubuntu des versions de Tanzu Kubernetes.

Attention La bibliothèque de contenu TKr est utilisée dans tous les espaces de noms vSphere du superviseur lorsque vous provisionnez de nouveaux clusters TKG.

Conditions préalables

En tant qu'administrateur de cloud, vérifiez que VMware Private AI Foundation with NVIDIA est déployé et configuré. Reportez-vous à la section Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI.

Procédure

1. Téléchargez les images TKr basées sur Ubuntu avec les versions de Kubernetes requises à partir de `https://wp-content.vmware.com/v2/latest/`.
2. Connectez-vous à l'instance de vCenter Server pour le domaine de charge de travail VI à l'adresse **`http://<vcenter_server_fqdn>/ui`**.
3. Créez une bibliothèque de contenu locale et importez-y les images TKr.

   Reportez-vous à la section Créer une bibliothèque de contenu locale (pour le provisionnement de cluster isolé).
4. Ajoutez la bibliothèque de contenu au superviseur.
   a. Sélectionnez **Menu > Gestion de la charge de travail**.
   b. Accédez au superviseur pour les charges de travail d'IA.
   c. Dans l'onglet **Configurer**, sélectionnez **Général**.
   d. En regard de la propriété **Tanzu Kubernetes Grid Service**, cliquez sur **Modifier**.
   e. Sur la page **Général** qui s'affiche, développez **Tanzu Kubernetes Grid Service**, puis, en regard de **Bibliothèque de contenu**, cliquez sur **Modifier**.
   f. Sélectionnez la bibliothèque de contenu avec les images TKr et cliquez sur **OK**.

Configuration d'un registre Harbor privé dans VMware Private AI Foundation with NVIDIA

Vous pouvez utiliser Harbor en tant que service de superviseur comme registre local pour les images de conteneur à partir du catalogue NVIDIA NGC.

Note L'installation du service Harbor dans le superviseur nécessite une connexion Internet.

Pour utiliser l'intégration du registre Harbor au superviseur, vous pouvez suivre les approches de configuration suivantes :

- Utilisez un registre Harbor uniquement dans le superviseur du domaine de charge de travail avec GPU activé. Effectuez les tâches suivantes :

  a Activer Harbor en tant que service de superviseur.

  b Charger des images de conteneur AI dans un registre Harbor privé de VMware Private AI Foundation with NVIDIA

  Vous pouvez déconnecter votre environnement d'Internet et utiliser le service Harbor comme registre de conteneur local après avoir installé le service ou après l'avoir installé et téléchargé l'ensemble initial d'images de conteneur requises.

  Dans cette approche, vous devez télécharger manuellement des images de conteneur à partir du catalogue NVIDIA NGC vers une machine de l'environnement, puis les charger dans le registre.

- Créer un registre Harbor dans VMware Private AI Foundation with NVIDIA en tant que réplica d'un registre connecté.

  Un registre Harbor, exécuté en dehors de l'environnement VMware Private AI Foundation with NVIDIA, est toujours connecté à Internet. Le registre Harbor dans le superviseur pour le domaine de charge de travail avec GPU activé reçoit des images de conteneur de l'instance connectée à l'aide d'un mécanisme de proxy. Les principaux composants de l'instance de VMware Cloud Foundation restent ainsi isolés.

  Dans cette approche, des ressources supplémentaires sont requises pour le registre connecté.

Note Allouez suffisamment d'espace de stockage pour héberger les conteneurs NVIDIA NGC que vous prévoyez de déployer sur une VM à apprentissage profond ou sur un cluster TKG. Intégrez au moins trois versions de chaque conteneur dans l'espace de stockage.

Si la connexion à Internet lors de l'installation du service Harbor ou de la configuration d'un registre Harbor connecté ne convient pas à votre organisation, utilisez un registre de conteneur par un autre fournisseur.

Charger des images de conteneur AI dans un registre Harbor privé de VMware Private AI Foundation with NVIDIA

Dans un environnement déconnecté dans lequel vous utilisez un registre Harbor uniquement sur le superviseur prêt pour l'IA, vous devez charger manuellement les images de conteneur d'IA que vous prévoyez de déployer sur une VM à apprentissage profond ou un cluster TKG à partir du catalogue NVIDIA NGC vers Harbor.

Procédure

1. Sur les machines permettant d'accéder à NVIDIA NGC et à l'instance de VMware Cloud Foundation déconnectée, configurez le client Docker avec le certificat du registre Harbor.

   Reportez-vous à la section Configurer un client Docker avec un certificat de registre.

2. Connectez-vous à NVIDIA NGC.

   Utilisez le nom d'utilisateur réservé de $oauthtoken et collez la clé API dans le champ Mot de passe.

   ```
   docker login nvcr.io
   ```

3. Extrayez les images de conteneur requises sur la machine disposant d'un accès au catalogue NVIDIA NGC et enregistrez-les dans une archive.

   Par exemple, pour télécharger l'exemple d'image de conteneur CUDA, exécutez les commandes suivantes.

   ```
   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. Copiez l'archive sur la machine disposant d'un accès au registre de conteneur local.

5. Sur la machine ayant accès au registre de conteneur local, chargez l'image de conteneur.

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

6. Connectez-vous au registre Harbor.

   Par exemple, si le registre Harbor s'exécute à my-harbor-registry.example.com, exécutez les commandes suivantes.

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

7 Balisez l'image que vous souhaitez transférer vers le projet avec le même nom que l'espace de noms où vous souhaitez l'utiliser.

   Par exemple, pour baliser l'exemple d'image de conteneur CUDA comme la plus récente pour le projet my-private-ai-namespace sur le registre my-harbor-registry.example.com, exécutez la commande suivante.

```
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 Transférez les images de conteneur vers le registre Harbor.

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

Créer un registre Harbor dans VMware Private AI Foundation with NVIDIA en tant que réplica d'un registre connecté

Pour faciliter la mise à jour vers les dernières images dans le catalogue NVIDIA NGC, vous pouvez utiliser un registre Harbor dans un superviseur qui se trouve dans un autre domaine de charge de travail VI ou une autre instance de VMware Cloud Foundation et qui peut être connecté à Internet. Répliquez ensuite ce registre connecté sur le superviseur sur lequel vous prévoyez d'exécuter des charges de travail d'IA.

Extrayez les dernières images du conteneur de NVIDIA NGC vers le registre Harbor connecté et transférez-les vers le registre déconnecté à l'aide d'une connexion mise en cache par proxy. Ainsi, vous n'avez pas besoin de télécharger des images de conteneur, puis de les charger manuellement fréquemment.

Note Vous pouvez également utiliser un registre de conteneur connecté par un autre fournisseur.

Configurez le réseau entre les deux registres de la manière suivante :

- Le registre connecté est routable vers le registre de réplicas.
- Le registre connecté est placé dans une zone DMZ dans laquelle seule la communication `docker push` et `docker pull` est autorisée entre les deux registres.

Conditions préalables

Activer Harbor en tant que service de superviseur dans le superviseur du domaine de charge de travail avec GPU activé.

Procédure

1 Connectez-vous à l'interface utilisateur du registre Harbor connecté en tant qu'administrateur système Harbor.

2 Accédez à la page **Administration > Registres** et créez un point de terminaison pour le catalogue NVIDIA NGC `nvcr.io/nvaie` en sélectionnant le fournisseur **Registre Docker** et avec votre clé API NVIDIA NGC.

3 Accédez à la page **Administration > Projets** et créez un projet proxy-cache, connecté au point de terminaison pour `nvcr.io/nvaie`.

4 En revenant sur la page **Registres**, créez un point de terminaison de réplication pour le registre déconnecté, en sélectionnant le fournisseur **Harbor**.

5 Accédez à la page **Administration > Réplications** et créez une règle de réplication.

- Utilisez le mode de réplication basé sur le transfert.
- Dans la propriété **Registre de destination**, entrez l'URL du registre déconnecté sur le superviseur prêt pour l'IA.
- Définissez les filtres, l'espace de noms cible et le mode de déclenchement en fonction des besoins de votre organisation.

Étape suivante

1 Extrayez les images de conteneur requises par votre organisation de NVIDIA NGC vers le registre connecté en exécutant `docker pull` sur la machine cliente Docker.

2 Si la règle de réplication dispose du mode de déclenchement manuel, exécutez les réplications manuellement si nécessaire.

Charger les composants de l'opérateur NVIDIA GPU dans un environnement déconnecté

Dans un environnement déconnecté, chargez les composants de l'opérateur NVIDIA GPU sur des emplacements internes.

Procédure

1 Fournissez un référentiel de modules Ubuntu local et chargez les images de conteneur du module opérateur NVIDIA GPU dans le registre Harbor pour le superviseur.

2 Fournissez un référentiel local de graphiques Helm avec des définitions de graphiques de l'opérateur NVIDIA GPU.

3 Mettez à jour les définitions des graphiques Helm de l'opérateur NVIDIA GPU pour utiliser le référentiel de modules Ubuntu local et le registre Harbor privé.

Résultats

Pour plus d'informations, reportez-vous à la section Installation de VMware vSphere with VMware Tanzu (isolé).

Configurer VMware Aria Automation pour VMware Private AI Foundation with NVIDIA

VMware Aria Automation prend en charge les éléments du catalogue en libre-service que les ingénieurs DevOps et les scientifiques des données peuvent utiliser pour provisionner des charges de travail d'IA dans VMware Private AI Foundation with NVIDIA de manière conviviale et personnalisable.

Conditions préalables

En tant qu'administrateur de cloud, vérifiez que l'environnement VMware Private AI Foundation with NVIDIA est configuré. Reportez-vous à la section Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI.

Procédure

1. Connecter VMware Aria Automation à un domaine de charge de travail pour VMware Private AI Foundation with NVIDIA

   Avant d'ajouter les éléments du catalogue pour le provisionnement d'applications d'IA à l'aide de VMware Aria Automation, connectez VMware Aria Automation à VMware Cloud Foundation.

2. Créer des éléments de catalogue en libre-service d'IA dans VMware Aria Automation

   En tant qu'administrateur de cloud, utilisez l'assistant de configuration de catalogue pour Private AI dans VMware Aria Automation afin d'ajouter rapidement des éléments de catalogue pour le déploiement de machines virtuelles à apprentissage profond ou de clusters TKG accélérés par GPU dans un domaine de charge de travail VI de l'instance de VMware Cloud Foundation connectée.

3. Créer un élément de catalogue de base de données vectorielle dans VMware Aria Automation

   En tant qu'administrateur de cloud, vous pouvez ajouter un élément de catalogue à Automation Service Broker dans VMware Aria Automation pour le provisionnement de bases de données dans VMware Data Services Manager.

Connecter VMware Aria Automation à un domaine de charge de travail pour VMware Private AI Foundation with NVIDIA

Avant d'ajouter les éléments du catalogue pour le provisionnement d'applications d'IA à l'aide de VMware Aria Automation, connectez VMware Aria Automation à VMware Cloud Foundation.

Procédure

- Dans l'interface utilisateur de VMware Aria Automation, exécutez l'assistant de démarrage rapide pour VMware Cloud Foundation ou pour VMware vCenter Server.

  Reportez-vous à la section Prise en main de VMware Aria Automation à l'aide du démarrage rapide de VMware Cloud Foundation ou Prise en main de VMware Aria Automation à l'aide du démarrage rapide de VMware vCenter Server dans la documentation *Prise en main de VMware Aria Automation*.

Créer des éléments de catalogue en libre-service d'IA dans VMware Aria Automation

En tant qu'administrateur de cloud, utilisez l'assistant de configuration de catalogue pour Private AI dans VMware Aria Automation afin d'ajouter rapidement des éléments de catalogue pour le déploiement de machines virtuelles à apprentissage profond ou de clusters TKG accélérés par GPU dans un domaine de charge de travail VI de l'instance de VMware Cloud Foundation connectée.

Les scientifiques de données peuvent utiliser des éléments de catalogue d'apprentissage profond pour le déploiement de VM à apprentissage profond. Les ingénieurs DevOps peuvent utiliser les éléments du catalogue pour le provisionnement de clusters TKG prêts pour l'IA.

Chaque fois que vous l'exécutez, l'assistant de configuration du catalogue pour Private AI ajoute des éléments pour les machines virtuelles à apprentissage profond et les clusters TKG au catalogue Service Broker. Vous pouvez exécuter l'assistant dans les cas suivants :

- Activation du provisionnement de charges de travail d'IA sur un autre superviseur.
- Intégration d'une modification de votre licence NVIDIA AI Enterprise, y compris le fichier `.tok` de la configuration client et le dispositif License Server, ou l'URL de téléchargement des pilotes invités vGPU pour un environnement déconnecté.
- Intégration d'une modification d'image de VM à apprentissage profond.
- Utilisation d'autres classes de VM vGPU ou sans GPU, d'une stratégie de stockage ou d'un registre de conteneur.
- Création d'éléments de catalogue dans un nouveau projet.

Note VMware Aria Automation crée un espace de noms vSphere lorsqu'une VM à apprentissage profond ou un cluster Tanzu Kubernetes Grid est provisionné.

Procédure

- Ajouter des éléments Private AI au catalogue Automation Service Broker.

Étape suivante

À l'aide d'Automation Service Broker, vos scientifiques des données et vos ingénieurs DevOps peuvent procéder au déploiement de VM à apprentissage profond, avec le provisionnement de

clusters Tanzu Kubernetes Grid prenant en charge les GPU. Reportez-vous à la section Déployer une machine virtuelle à apprentissage profond sans RAG dans VMware Aria Automation.

Créer un élément de catalogue de base de données vectorielle dans VMware Aria Automation

En tant qu'administrateur de cloud, vous pouvez ajouter un élément de catalogue à Automation Service Broker dans VMware Aria Automation pour le provisionnement de bases de données dans VMware Data Services Manager.

Conditions préalables

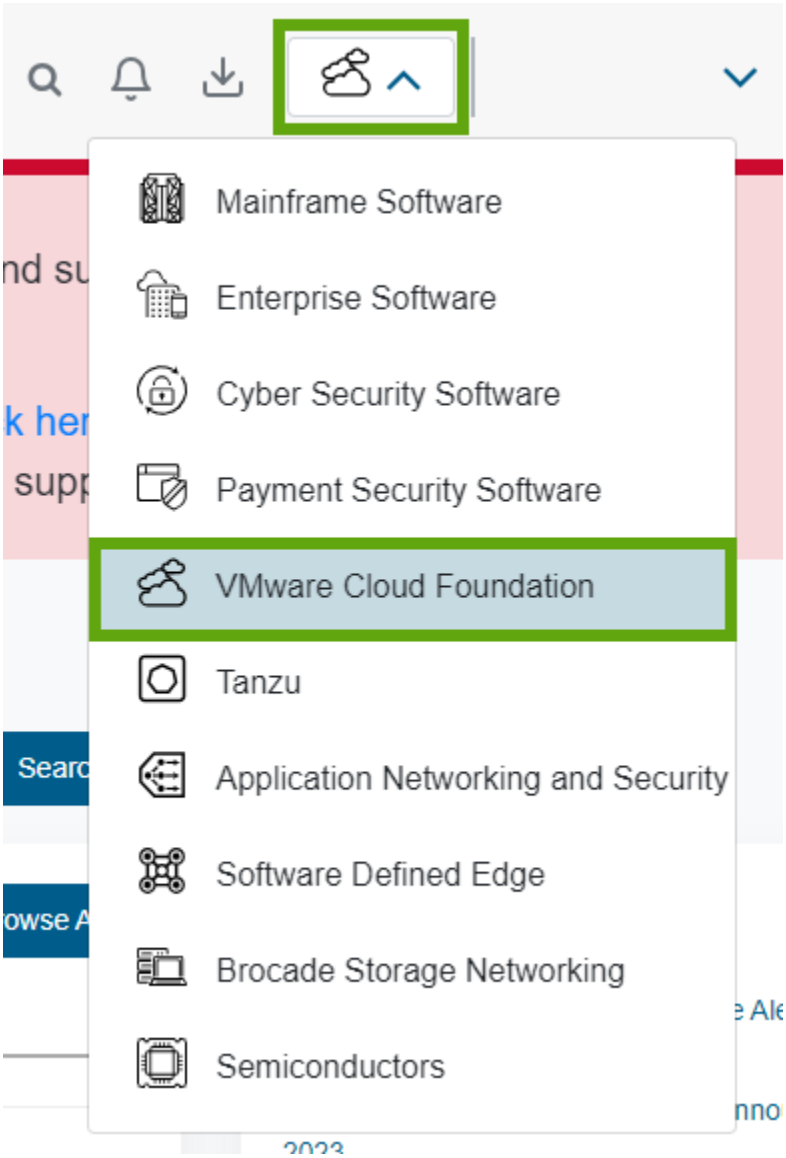
- Vérifiez que VMware Data Services Manager 2.1 est déployé.
- Fournissez une machine sur laquelle Python 3.10 est installé et qui a accès aux instances de VMware Data Services Manager et de VMware Aria Automation.

Procédure

1 Téléchargez le bundle `AriaAutomation_DataServicesManager` pour VMware Data Services Manager 2.1 à partir du portail technique broadcom.

   a Connectez-vous au portail de support Broadcom.

   b Dans le menu déroulant Catégorie de logiciel situé dans le coin supérieur droit du portail, sélectionnez **VMware Cloud Foundation**.



   c Dans le volet de navigation de gauche, cliquez sur **Mes téléchargements**.

   d Sur la page **Mes téléchargements - VMware Cloud Foundation**, cliquez sur **VMware Data Services Manager**.

   e Cliquez sur le numéro de version et téléchargez le bundle `AriaAutomation_DataServicesManager`.

2 Sur la machine exécutant Python, chargez le bundle `AriaAutomation_DataServicesManager` et extrayez son contenu.

3 Mettez à jour le fichier `config.json` dans le dossier dans lequel vous avez extrait le bundle avec les URL et les informations d'identification de l'utilisateur pour VMware Data Services Manager et VMware Aria Automation.

   Vous pouvez éventuellement définir aussi le nom de l'élément du catalogue, le projet Automation Assembler et d'autres paramètres.

4 Pour créer les éléments du catalogue dans VMware Aria Automation, exécutez le script Python `aria.py` de la manière suivante.

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

Résultats

Le script Python crée des éléments dans VMware Aria Automation requis afin d'utiliser VMware Data Services Manager pour le provisionnement de bases de données. Reportez-vous au fichier `readme.md` dans le bundle `AriaAutomation_DataServicesManager`

Étape suivante

Vos scientifiques des données ou ingénieurs DevOps peuvent déployer une base de données vectorielle à partir du catalogue Automation Service Broker avec l'extension pgvector et l'intégrer à leurs charges de travail RAG. Reportez-vous à la section Chapitre 5 Déploiement de charges de travail RAG dans VMware Private AI Foundation with NVIDIA.

Déploiement d'une VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA

3

VMware Private AI Foundation with NVIDIA prend en charge le provisionnement de VM à apprentissage profond préconfigurées que les scientifiques des données peuvent utiliser pour le développement de l'IA.

En tant que scientifique des données, vous disposez des options suivantes pour commencer à utiliser une VM à apprentissage profond :

- Déployer une VM à apprentissage profond à l'aide d'un élément de catalogue en libre-service dans VMware Aria Automation.
- Demander à votre ingénieur DevOps de déployer une VM à apprentissage profond sur un cluster Tanzu Kubernetes Grid à l'aide de la commande `kubectl`.
- Demander à votre administrateur de cloud de déployer une VM à apprentissage profond sur un cluster vSphere pour découvrir rapidement les modèles de VM à apprentissage profond.
- À propos des images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA

  Les images de machine virtuelle à apprentissage profond fournies dans le cadre de VMware Private AI Foundation with NVIDIA sont préconfigurées avec les bibliothèques ML, infrastructures et kit d'outils populaires, et sont optimisées et validées par NVIDIA et VMware pour l'accélération par GPU dans un environnement VMware Cloud Foundation.
- Déployer une VM à apprentissage profond à l'aide d'un catalogue en libre-service dans VMware Aria Automation

  Dans VMware Private AI Foundation with NVIDIA, en tant que scientifique des données ou ingénieur DevOps, vous pouvez déployer une VM à apprentissage profond à partir de VMware Aria Automation en utilisant des éléments de catalogue de poste de travail en libre-service d'IA dans Automation Service Broker.
- Déployer une VM à apprentissage profond directement sur un cluster vSphere de VMware Private AI Foundation with NVIDIA

  En tant qu'administrateur de cloud, pour permettre rapidement aux scientifiques des données de tester les modèles de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA, vous pouvez déployer une VM à apprentissage profond directement sur un cluster vSphere à l'aide de vSphere Client.

- Déployer une VM à apprentissage profond à l'aide de la commande `kubectl` dans VMware Private AI Foundation with NVIDIA

  Le service de VM dans le superviseur de vSphere IaaS Control Plane permet aux ingénieurs DevOps de déployer et d'exécuter des VM à apprentissage profond à l'aide de l'API Kubernetes.

- Personnalisation du déploiement de VM en apprentissage profond dans VMware Private AI Foundation with NVIDIA

  Lorsque vous déployez une VM à apprentissage profond dans vSphere IaaS control plane à l'aide de `kubectl` ou directement sur un cluster vSphere, vous devez renseigner les propriétés personnalisées de la VM.

- Dépannage du déploiement de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA

  Les informations de dépannage sur le déploiement d'une VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA fournissent des solutions aux problèmes potentiels que vous pourriez rencontrer.

À propos des images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA

Les images de machine virtuelle à apprentissage profond fournies dans le cadre de VMware Private AI Foundation with NVIDIA sont préconfigurées avec les bibliothèques ML, infrastructures et kit d'outils populaires, et sont optimisées et validées par NVIDIA et VMware pour l'accélération par GPU dans un environnement VMware Cloud Foundation.

En tant que scientifique des données, vous pouvez utiliser les VM à apprentissage profond provisionnées à partir de ces images pour le prototypage, le réglage fin, la validation et l'inférence de l'IA.

La pile logicielle pour l'exécution d'applications d'IA au-dessus des GPU NVIDIA est validée à l'avance. Par conséquent, démarrez directement le développement de l'IA, sans passer de temps à installer et à valider la compatibilité des systèmes d'exploitation, des bibliothèques logicielles, des infrastructures ML, des kits d'outils et des pilotes GPU.

Que contient une image de VM à apprentissage profond ?

La dernière image de machine virtuelle à apprentissage profond contient le logiciel suivant. Pour plus d'informations sur les versions des composants dans chaque version d'image de VM à apprentissage profond, reportez-vous à la section Notes de mise à jour de VMware Deep Learning VM.

Catégorie de composant logiciel	Composant logiciel	
Intégrée	■ Canonical Ubuntu ■ NVIDIA Container Toolkit ■ Docker Community Engine ■ Miniconda et un manifeste Conda pour PyTorch.	
Vous pouvez les préinstaller automatiquement lorsque vous démarrez la VM à apprentissage profond pour la première fois	■ Pilote invité vGPU, selon la version du pilote d'hôte vGPU	
	Charges de travail à apprentissage profond (DL,Deep learning)	Exemple CUDA Vous pouvez utiliser une VM à apprentissage profond avec des exemples CUDA en cours d'exécution pour explorer l'ajout de vecteurs, la simulation gravitationnelle à N corps ou d'autres exemples sur une VM. Reportez-vous à la page Exemples de CUDA du catalogue NVIDIA NGC.
		PyTorch. Vous pouvez utiliser une VM à apprentissage profond avec une bibliothèque PyTorch pour découvrir l'IA conversationnelle, le traitement automatique des langues (NLP, Natural language processing) et d'autres types de modèles d'IA sur une VM. Reportez-vous à la page PyTorch du catalogue NVIDIA NGC. Vous pouvez utiliser une instance de JupyterLab prête avec PyTorch installé et configuré à l'adresse `http://dl_vm_ip:8888`.
		TensorFlow. Vous pouvez utiliser une VM à apprentissage profond avec une bibliothèque TensorFlow pour découvrir l'IA conversationnelle, le NLP et d'autres types de modèles d'IA sur une VM. Reportez-vous à la page TensorFlow du catalogue NVIDIA NGC. Vous pouvez utiliser une instance de JupyterLab prête avec TensorFlow installé et configuré à l'adresse `http://dl_vm_ip:8888`.
		Exportateur DCGM Vous pouvez utiliser une VM à apprentissage profond avec l'exportateur Data Center GPU Manager (DCGM) pour surveiller la santé des GPU et obtenir des mesures de ceux-ci qui sont utilisés par une charge de travail DL, à l'aide de NVIDIA DCGM, Prometheus et Grafana. Reportez-vous à la page Exportateur DCGM du catalogue NVIDIA NGC. Dans une VM à apprentissage profond, exécutez le conteneur de l'exportateur DCGM avec une charge de travail DL qui effectue des opérations d'IA. Une fois la VM à apprentissage profond démarrée, l'exportateur DCGM est prêt à collecter des mesures de vGPU et à exporter les données vers une autre application pour une surveillance et une visualisation accrues. Pour plus d'informations sur l'utilisation de l'exportateur DGCM pour visualiser les mesures avec Prometheus et Grafana, reportez-vous à la section Exportateur DCGM.

Catégorie de composant logiciel	Composant logiciel	
		Serveur d'inférence Triton Vous pouvez utiliser une VM à apprentissage profond avec un serveur d'inférence Triton pour charger un référentiel de modèles et recevoir des demandes d'inférence. Reportez-vous à la page Serveur d'inférence Triton du catalogue NVIDIA NGC. Pour plus d'informations sur l'utilisation du serveur d'inférence Triton pour les demandes d'inférence pour les modèles d'IA, reportez-vous à la section Serveur d'inférence Triton.
		NVIDIA RAG Vous pouvez utiliser une VM à apprentissage profond pour créer des solutions de génération augmentée de récupération (RAG) avec un modèle Llama2. Reportez-vous à la documentation Outil Docker Compose des applications NVIDIA RAG (nécessite des autorisations de compte spécifiques). Exemple d'application Web d'agent conversationnel accessible à l'adresse `http://dl_vm_ip:3001/orgs/nvidia/models/text-qa-chatbot`. Vous pouvez charger votre propre base de connaissances.

Déploiement d'une VM à apprentissage profond

En tant que scientifique des données, vous pouvez déployer vous-même une VM à apprentissage profond en utilisant des éléments de catalogue dans VMware Aria Automation. Sinon, un administrateur de cloud ou un ingénieur DevOps déploie cette VM pour vous.

Déployer une VM à apprentissage profond à l'aide d'un catalogue en libre-service dans VMware Aria Automation

Dans VMware Private AI Foundation with NVIDIA, en tant que scientifique des données ou ingénieur DevOps, vous pouvez déployer une VM à apprentissage profond à partir de VMware Aria Automation en utilisant des éléments de catalogue de poste de travail en libre-service d'IA dans Automation Service Broker.

Pour plus d'informations sur les images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA, reportez-vous à la section À propos des images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA.

Conditions préalables

- Vérifiez que votre administrateur de cloud a configuré le catalogue VMware Aria Automation pour le déploiement d'applications Private AI. Reportez-vous à la section Ajouter des éléments Private AI au catalogue Automation Service Broker.

- Vérifiez que votre administrateur de cloud a attribué le rôle d'utilisateur requis pour le déploiement de VM à apprentissage profond.

Procédure

- Déployer une machine virtuelle à apprentissage profond sans RAG dans VMware Aria Automation ou Déployer une VM à apprentissage profond avec une charge de travail RAG.

  Le déploiement d'une VM à apprentissage profond avec NVIDIA RAG nécessite une base de données vectorielle, telle qu'une base de données PostgreSQL avec pgvector dans VMware Data Services Manager.

Résultats

Le pilote invité vGPU et la charge de travail d'apprentissage profond spécifiée sont installés lorsque vous démarrez la VM à apprentissage profond.

Étape suivante

Pour plus d'informations sur l'accès à la machine virtuelle et à l'instance de JupyterLab fournie avec certaines images de la VM à apprentissage profond, accédez à **Consommer > Déploiements > Déploiements** dans Automation Service Broker.

Déployer une VM à apprentissage profond directement sur un cluster vSphere de VMware Private AI Foundation with NVIDIA

En tant qu'administrateur de cloud, pour permettre rapidement aux scientifiques des données de tester les modèles de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA, vous pouvez déployer une VM à apprentissage profond directement sur un cluster vSphere à l'aide de vSphere Client.

Pour plus d'informations sur les images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA, reportez-vous à la section À propos des images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA.

Le déploiement d'une VM à apprentissage profond avec NVIDIA RAG nécessite une base de données vectorielle, telle qu'une base de données PostgreSQL avec pgvector dans VMware Data Services Manager. Pour plus d'informations sur le déploiement d'une base de données de ce type et son intégration dans une VM à apprentissage profond, reportez-vous à la section Déployer une VM à apprentissage profond avec une charge de travail RAG.

Conditions préalables

Vérifiez que VMware Private AI Foundation with NVIDIA est déployé et configuré. Reportez-vous à la section Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI.

Procédure

1. Connectez-vous à l'instance de vCenter Server pour le domaine de charge de travail VI.
2. Dans le menu d'accueil de vSphere Client, sélectionnez **Bibliothèques de contenu**.
3. Accédez à l'image de VM à apprentissage profond dans la bibliothèque de contenu.
4. Cliquez avec le bouton droit sur un modèle OVF et sélectionnez **Nouvelle VM à partir de ce modèle**.
5. Sur la page **Sélectionner un nom et un dossier** de l'assistant qui s'affiche, entrez un nom et sélectionnez un dossier de VM, sélectionnez **Personnaliser le matériel de cette machine virtuelle**, puis cliquez sur **Suivant**.
6. Sélectionnez un cluster prenant en charge les GPU dans le domaine de charge de travail VI, indiquez si la machine virtuelle doit être mise sous tension une fois le déploiement terminé, puis cliquez sur **Suivant**.
7. Suivez l'assistant pour sélectionner une banque de données et un réseau sur le commutateur Distributed Switch pour le cluster.
8. Sur la page **Personnaliser le modèle**, entrez les propriétés de VM personnalisées requises pour configurer la fonctionnalité d'IA, puis cliquez sur **Suivant**.

   Reportez-vous à la section Propriétés OVF des VM à apprentissage profond.
9. Sur la page **Personnaliser le matériel**, attribuez un périphérique NVIDIA vGPU à la machine virtuelle en tant que **Nouveau périphérique PCI**, puis cliquez sur **Suivant**.

   Pour une VM à apprentissage profond qui exécute un dispositif NVIDIA RAG, sélectionnez le profil vGPU complet pour le mode de découpage temporel ou un profil MIG. Par exemple, pour le dispositif NVIDIA A100 40 Go en mode de découpage temporel vGPU, sélectionnez **nvidia_a100-40c**.
10. Pour une VM à apprentissage profond qui exécute un dispositif NVIDIA RAG, définissez le paramètre `pciPassthru<vgpu-id>.cfg.enable_uvm` sur **1** dans l'onglet **Paramètres avancés** des paramètres de la machine virtuelle.

    où `<vgpu-id>` identifie le vGPU attribué à la machine virtuelle. Par exemple, si deux vGPU sont attribués à la machine virtuelle, définissez `pciPassthru0.cfg.parameter=1` et `pciPassthru1.cfg.parameter = 1`.
11. Examinez les spécifications du déploiement et cliquez sur **Terminer**.

Résultats

Le pilote invité vGPU et la charge de travail d'apprentissage profond spécifiée sont installés lorsque vous démarrez la VM à apprentissage profond.

Vous pouvez examiner les journaux ou ouvrir l'instance de JupyterLab fournie avec certaines images. Vous pouvez partager les détails de l'accès avec des scientifiques des données de votre organisation. Reportez-vous à la section Charges de travail d'apprentissage profond dans VMware Private AI Foundation with NVIDIA.

Étape suivante

- Connectez-vous à la VM à apprentissage profond via SSH et vérifiez que tous les composants sont installés et en cours d'exécution comme prévu.
- Envoyez les détails d'accès à vos scientifiques des données.

Déployer une VM à apprentissage profond à l'aide de la commande `kubectl` dans VMware Private AI Foundation with NVIDIA

Le service de VM dans le superviseur de vSphere IaaS Control Plane permet aux ingénieurs DevOps de déployer et d'exécuter des VM à apprentissage profond à l'aide de l'API Kubernetes.

En tant qu'ingénieur DevOps, utilisez `kubectl` pour déployer une VM à apprentissage profond sur l'espace de noms configuré par l'administrateur de cloud.

Pour plus d'informations sur les images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA, reportez-vous à la section À propos des images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA.

Le déploiement d'une VM à apprentissage profond avec NVIDIA RAG nécessite une base de données vectorielle, telle qu'une base de données PostgreSQL avec pgvector dans VMware Data Services Manager. Pour plus d'informations sur le déploiement d'une base de données de ce type et son intégration dans une VM à apprentissage profond, reportez-vous à la section Déployer une VM à apprentissage profond avec une charge de travail RAG.

Conditions préalables

Vérifiez auprès de l'administrateur de cloud que VMware Private AI Foundation with NVIDIA est déployé et configuré. Reportez-vous à la section Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI.

Procédure

1 Connectez-vous au plan de contrôle du superviseur.

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

2 Vérifiez que toutes les ressources de VM requises, telles que les classes de VM et les images de VM, sont en place sur l'espace de noms.

Reportez-vous à la section Afficher les ressources de VM disponibles sur un espace de noms dans vSphere with Tanzu.

3 Préparez le fichier YAML pour la VM à apprentissage profond.

Utilisez la valeur vm-operator-api, en définissant les propriétés OVF comme objet ConfigMap. Pour plus d'informations sur les propriétés OVF disponibles, reportez-vous à la section Propriétés OVF des VM à apprentissage profond.

Par exemple, vous pouvez créer une spécification YAML `example-dl-vm.yaml` à titre d'exemple de VM à apprentissage profond exécutant PyTorch dans un environnement connecté.

```
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
RoZSBVUkkgcGFydAogICAgaWYgW1sgJFJFR0lTVFJZX1VSSV9QQVRIID09ICIoLyIqIF1dOyB0aGVuCiAgICAgIFJFR
0lTVFJZX1VSST0kKGVjaG8gIiRSRUdJU1RSWV9VUklfUEFUSCIgfCBjdXQgLWQnLycgLWYxKQogICAgZWxzZQogICAg
ICBSRUdJU1RSWV9VUkk9JFJFR0lTVFJZX1VSSV9QQVRICiAgICBmaQogIAogICAgUkVHSVNUUllfVVNFUk5BTUU9JCh
ncmVwIHJlZ2lzdHJ5LXVzZXIgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW1
4iXSpcKS4qL1wxL3AnKQogICAgUkVHSVNUUllfUEFTU1dPUkQ9JChncmVwIHJlZ2lzdHJ5LXBhc3N3ZCAvb3B0L2Rsd
m0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCiAgICBpZiBbWyAt
biAiJFJFR0lTVFJZX1VTRVJOQU1FIiAmJiAtbiAiJFJFR0lTVFJZX1BBU1NXT1JEIiBdXTsgdGhlbgogICAgICBkb2N
rZXIgbG9naW4gLXUgJFJFR0lTVFJZX1VTRVJOQU1FIC1wICRSRUdJU1RSWV9QQVNTV09SRCAkUkVHSVNUUllfVVJJCi
AgICBlbHNlCiAgICAgIGVjaG8gIldhcm5pbmc6IHRoZSByZWdpc3RyeSdzIHVzZXJuYW1lIGFuZCBwYXNzd29yZCBhc
mUgaW52YWxpZCwgU2tpcHBpbmcgRG9ja2VyIGxvZ2luLiIKICAgIGZpCgogICAgZG9ja2VyIHJ1biAtZCAtLWdwdXMg
YWxsIC1wIDg4ODg6ODg4OCAkUkVHSVNUUllfVVJJX1BBVEgvbnZpZGlhL3B5dG9yY2g6MjMuMTAtcHkzIC91c3IvbG9
jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9vdCAtLWlwPSogLS1wb3J0PTg4ODggLS1uby1icm93c2VyIC0tTm
90ZWJvb2tBcHAudG9rZW49JycgLS1ob3RlYm9va0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGlyPS93b
3Jrc3BhY2UKCi0gcGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6
IHwKICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAgICBlY2hvICJFcnJvcjogJDEiID4mMgogICA
gICB2bXRvb2xzZCAtLWNtZCAiaW5mby1zZXQgZ3Vlc3RpbmZvLnZtc2Vydmljj2S5ib290c3RyYXAuY29uZGl0aW9uIG
ZhbHNlLCBETFdvcmtsb2FkRmFpbHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgICAgfQoKICAgIGNoZWNrX3Byb3RvY29sK
```

```
CkgewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQKICAgICAgbG9jYWwgc3VwcG9ydGVkX3Byb3Rv
Y29scz0oIiRAIikKICAgICAgaWYgW1sgLW4gIiR7cHJveHlfdXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm9
0b2NvbD0kKGVjaG8gIiR7cHJveHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBwcmludCAkMTsgZW
xzZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2NvbCIgXTsgdGhlbgogICAgICAgICAgZWNobyAiT
m8gc3BlY2lmaWMgcHJvdG9jb2wgcHJvdmlkZWQuIFNraXBwaW5nIHByb3RvY29sIGNoZWNrLiIKICAgICAgICAgIHJl
dHVybiAwCiAgICAgICAgZmkKICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQogICAgICAgIGZvciB
2YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwogICAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2x9Ii
A9PSAiJHt2YXJ9IiBdXTsgdGhlbgogICAgICAgICAgICBwcm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgI
GJyZWFrCiAgICAgICAgICBmaQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2NvbF9pbmNsdWRlZH0i
ID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0ICJVbnN1cHBvcnRlZCBwcm90b2NvbDogJHtwcm9
0b2NvbH0uIFN1cHBvcnRlZCBwcm90b2NvbHMgYXJlOiAke3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgIG
ZpCiAgICAgIGZpCiAgICB9CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xzCiAgICBzZXRfcHJve
HkoKSB7CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIpCgogICAgICBDT05GSUdfSlNPTl9CQVNF
NjQ9JChncmVwICdjb25maWctanNvbicgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU
9IlwoW14iXSpcKS4qL1wxL3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdfSlNPTl9CQVNFNjR9IH
wgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgSFRUUF9QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqc
SAtciAnLmh0dHBfcHJveHkgLy8gZW1wdHknKQogICAgICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19K
U09OfSIgfCBqcSAtciAnLmh0dHBzX3Byb3h5IC8vIGVtcHR5JykKICAgICAgaWYgW1sgJD8gLW5lIDAgfHwgKC16ICI
ke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJPWFlfVVJMfSIpIF1dOyB0aGVuCiAgICAgICAgZWNoby
AiSW5mbzogVGhlIGNvbmZpZy1qc29uIHdhcyBwYXJzZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kL
iIKICAgICAgICByZXR1cm4gMAogICAgICBmaQoKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUF9QUk9YWV9VUkx9
IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJ
MfSIgIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iCgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldG
MvZW52aXJvbm1lbnQ7IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0ke0hUVFBfUFJPWFlfVVJMf
QogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBTX1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSFRUUF9Q
Uk9YWT0ke0hUVFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH0
KICAgICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0Yy9lbnZpcm9ubWVudAogIC
AgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAgICAgIGZpCiAgICAgIAogICAgICAjIENvbmZpZ3VyZSBEb2NrZ
XIgdG8gdXNlIGEgcHJveHkKICAgICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5k
CiAgICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJPWFk9JHtIVFRQX1BST1hZX1V
STH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BST1hZPSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgIEVudm
lyb25tZW50PVwiTk9fUFJPWFk9bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9ja
2VyLnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVsb2FkCiAgICAgIHN5c3RlbWN0
bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAiSW5mbzogZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJ
```

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

Note `user-data` est la valeur codée en base64 pour le code cloud-init suivant :

```
#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 Basculez vers le contexte de l'espace de noms vSphere créé par l'administrateur de cloud.

Par exemple, pour un espace de noms appelé `example-dl-vm-namespace` :

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

5 Déployez la VM à apprentissage profond.

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

6 Pour vérifier que la VM a été créée, exécutez les commandes suivantes.

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

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

7 Exécutez un ping sur l'adresse IP de la machine virtuelle attribuée par le service de mise en réseau demandé.

Pour obtenir l'adresse publique et les ports afin d'accéder à la VM à apprentissage profond, obtenez les détails sur le service d'équilibrage de charge créé.

```
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
```

Résultats

Le pilote invité vGPU et la charge de travail DL spécifiée sont installés lorsque vous démarrez la VM à apprentissage profond.

Étape suivante

- Vous pouvez examiner les journaux ou ouvrir le bloc-notes JupyterLab fourni avec certaines des images. Reportez-vous à la section Charges de travail d'apprentissage profond dans VMware Private AI Foundation with NVIDIA.
- Envoyez les détails d'accès à vos scientifiques des données.

Personnalisation du déploiement de VM en apprentissage profond dans VMware Private AI Foundation with NVIDIA

Lorsque vous déployez une VM à apprentissage profond dans vSphere IaaS control plane à l'aide de `kubectl` ou directement sur un cluster vSphere, vous devez renseigner les propriétés personnalisées de la VM.

Pour plus d'informations sur les images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA, reportez-vous à la section À propos des images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA.

Propriétés OVF des VM à apprentissage profond

Lorsque vous déployez une VM à apprentissage profond, vous devez remplir des propriétés de VM personnalisées pour automatiser la configuration du système d'exploitation Linux, le déploiement du pilote invité vGPU, ainsi que le déploiement et la configuration de conteneurs NGC pour les charges de travail DL.

La dernière image de VM à apprentissage profond dispose des propriétés OVF suivantes :

Catégorie	Paramètre	Étiquette dans vSphere Client	Description
Propriétés du système d'exploitation de base	instance-id	ID de l'instance	Requis. ID d'instance unique de l'instance de VM. Un ID d'instance identifie de manière unique une instance. Lorsqu'un ID d'instance change, cloud-init traite l'instance comme une nouvelle instance et réexécute le processus cloud-init.
	hostname	Nom d'hôte	Requis. Nom d'hôte du dispositif.
	seedfrom	URL à partir de laquelle amorcer les données de l'instance	Facultatif. URL à partir de laquelle extraire la valeur du paramètre user-data et les métadonnées.
	public-keys	Clé publique SSH	Si ce paramètre est fourni, l'instance renseigne le `authorized_keys` SSH de l'utilisateur par défaut avec cette valeur.

Catégorie	Paramètre	Étiquette dans vSphere Client	Description
	user-data	Paramètre user-data codé	Ensemble de scripts ou d'autres métadonnées qui sont insérés dans la VM lors du provisionnement. Cette propriété est le contenu réel du script `cloud-init`. Cette valeur doit être codée en base64. ■ Vous pouvez utiliser cette propriété pour spécifier le conteneur de charges de travail DL à déployer, tel que PyTorch ou TensorFlow. Reportez-vous à la section Charges de travail d'apprentissage profond dans VMware Private AI Foundation with NVIDIA. ■ Utilisez cette propriété pour définir une adresse IP statique sur une machine virtuelle déployée directement sur un cluster vSphere. Reportez-vous à la section Attribuer une adresse IP statique à une VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA.
	password	Mot de passe de l'utilisateur par défaut	Requis. Mot de passe du compte d'utilisateur **vmware** par défaut.
Installation du pilote vGPU	vgpu-license	Licence vGPU	Requis. Jeton de configuration du client NVIDIA vGPU. Le jeton est enregistré dans le fichier `/etc/nvidia/ClientConfigToken/client_configuration_token.tok`.
	nvidia-portal-api-key	Clé API du portail NVIDIA	Requis dans un environnement connecté. Clé API que vous avez téléchargée à partir du portail de licences NVIDIA. La clé est requise pour l'installation du pilote invité vGPU.
	vgpu-fallback-version	Version du pilote d'hôte vGPU	Installez directement cette version du pilote invité vGPU.
	vgpu-url	URL pour les téléchargements de vGPU isolés	Requis dans un environnement déconnecté. URL à partir de laquelle télécharger le pilote invité vGPU. Pour plus d'informations sur la configuration requise du serveur Web local, reportez-vous à la section Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI.
Automatisation de la charge de travail DL	registry-uri	URI du registre	Requis dans un environnement déconnecté ou si vous prévoyez d'utiliser un registre de conteneur privé pour éviter de télécharger des images depuis Internet. URI d'un registre de conteneur privé avec les images de conteneur de charges de travail d'apprentissage profond. Requis si vous faites référence à un registre privé dans `user-data` ou `image-oneliner`.
	registry-user	Nom d'utilisateur du registre	Requis si vous utilisez un registre de conteneur privé qui nécessite une authentification de base.
	registry-passwd	Mot de passe du registre	Requis si vous utilisez un registre de conteneur privé qui nécessite une authentification de base.

Catégorie	Paramètre	Étiquette dans vSphere Client	Description
	registry-2-uri	URI du registre secondaire	Requis si vous utilisez un deuxième registre de conteneur privé basé sur Docker et s'il nécessite l'authentification de base. Par exemple, lors du déploiement d'une VM à apprentissage profond avec la charge de travail NVIDIA RAG DL préinstallée, une image pgvector est téléchargée à partir de Docker Hub. Vous pouvez utiliser les paramètres `registry-2-` pour contourner une limite de débit d'extraction pour docker.io.
	registry-2-user	Nom d'utilisateur du registre secondaire	Requis si vous utilisez un deuxième registre de conteneur privé.
	registry-2-passwd	Mot de passe du registre secondaire	Requis si vous utilisez un deuxième registre de conteneur privé.
	image-oneliner	Commande codée sur une ligne	Commande bash sur une ligne exécutée lors du provisionnement de VM. Cette valeur doit être codée en base64. Vous pouvez utiliser cette propriété pour spécifier le conteneur de charges de travail DL à déployer, tel que PyTorch ou TensorFlow. Reportez-vous à la section Charges de travail d'apprentissage profond dans VMware Private AI Foundation with NVIDIA. **Attention** Évitez d'utiliser `user-data` et `image-oneliner`.
	docker-compose-uri	Fichier Docker Compose codé	Requis si vous avez besoin d'un fichier Docker Compose pour démarrer le conteneur de charges de travail DL. Contenu du fichier `docker-compose.yaml` qui sera inséré dans la machine virtuelle lors du provisionnement après le démarrage de la machine virtuelle avec GPU activé. Cette valeur doit être codée en base64.
	config-json	Fichier config.json codé	Contenu d'un fichier de configuration pour ajouter des détails pour les serveurs proxy. Cette valeur doit être codée en base64. Reportez-vous à la section Configurer une VM à apprentissage profond avec un serveur proxy.
	conda-environment-install	Installation de l'environnement Conda	Liste séparée par des virgules d'environnements Conda à installer automatiquement à la fin du déploiement de VM. Environnements disponibles : `pytorch2.3_py3.12`

Charges de travail d'apprentissage profond dans VMware Private AI Foundation with NVIDIA

Vous pouvez provisionner une machine virtuelle à apprentissage profond avec une charge de travail d'apprentissage profond (DL) prise en charge en plus de ses composants intégrés. Les charges de travail DL sont téléchargées à partir du catalogue NVIDIA NGC et sont optimisées pour le GPU et validées par NVIDIA et VMware by Broadcom.

Pour obtenir une présentation des images de VM à apprentissage profond, reportez-vous à la section À propos des images de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA.

Exemple CUDA

Vous pouvez utiliser une VM à apprentissage profond avec des exemples CUDA en cours d'exécution pour explorer l'ajout de vecteurs, la simulation gravitationnelle à N corps ou d'autres exemples sur une VM. Reportez-vous à la page Exemples CUDA.

Une fois la VM à apprentissage profond lancée, elle exécute une charge de travail d'exemples CUDA pour tester le pilote invité vGPU. Vous pouvez examiner la sortie de test dans le fichier `/var/log/dl.log`.

Tableau 3-1. Image de conteneur d'exemples CUDA

Composant	Description
Image de conteneur	`nvcr.io/nvidia/k8s/cuda-sample:ngc_image_tag` Par exemple : `nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8` Pour plus d'informations sur les images du conteneur d'exemples CUDA qui sont prises en charge pour les VM à apprentissage profond, reportez-vous à la section Notes de mise à jour de VMware Deep Learning VM.
Entrées requises	Pour déployer une charge de travail d'exemples CUDA, vous devez définir les propriétés OVF de la machine virtuelle à apprentissage profond de la manière suivante : ■ Utilisez l'une des propriétés suivantes qui sont spécifiques à l'image d'exemples CUDA. ■ Script cloud-init. Codez-le au format base64.

```
#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
```

Tableau 3-1. Image de conteneur d'exemples CUDA (suite)

Composant	Description

```
- 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[@]}"
```

Tableau 3-1. Image de conteneur d'exemples CUDA (suite)

Composant	Description

```
      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"
    }
```

Par exemple, pour vectoradd-cuda11.7.1-ubi8, fournissez le script suivant au format base64 :

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICBz
ZXRfcHJveHkgImh0dHAiICJodHRwcyIgInNvY2tzNSIKICAgIHRyYXAgJ2Vycm9yX2V4
aXQgIlVuZXhwZWN0ZWQgZXJyb3Igb2NjdXJzIGF0IGRsIHdvcmtsb2FkIicgRVJSCiAg
ICBERUZBVUxUX1JFR19VUkk9Im52Y3IuaW8iCiAgICBSRUdJU1RSWV9VUklfUEFUSD0k
KGdyZXAgcmVnaXN0cnktdXJpIC9vcHQvZGx2bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAn
cy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKICAgICBpZiBbWyAteiAiJFJF
R0lTVFJZX1VSSV9QQVRIIiBdXTsgdGhlbgogICAgICAjIElmIFJFR0lTVFJZX1VSSV9Q
QVRIIGlzIG51bGwgb3IgZW1wdHksIHVzZSB0aGUgZGVmYXVsdCB2YWx1ZQogICAgICBS
RUdJU1RSWV9VUklfUEFUSD0kREVGQVVMVF9SRUdfVVJJCiAgICAgICAgIGVjaG8gIlJFR0lT
VFJZX1VSSV9QQVRIIHdhcyBlbXB0eS4gVXNpbmcgZGVmYXVsdDogJFJFR0lTVFJZX1VS
SV9QQVRIIgogICAgZmkKICAgICAgICAgIyBJZiBSRUdJU1RSWV9VUklfUEFUSCBjb250
YWlucyAnLycsIGV4dHJhY3QgdGhlIFVSSSBwYXJ0CiAgICBpZiBbWyAkUkVHSVNUUllf
VVJJX1BBVEggPT0gKiIvIiogXV07IHRoZW4KICAgICAgUkVHSVNUUllfVVJJPSQoZWNo
byAiJFJFR0lTVFJZX1VSSV9QQVRIIiB8IGN1dCAtZCcvJyAtZjEpCiAgICBlbHNlCiAg
ICAgIFJFR0lTVFJZX1VSST0kUkVHSVNUUllfVVJJX1BBVEgKICAgIGZpCiAgICBS
RUdJU1RSWV9VU0VSTkFNRT0kKGdyZXAgcmVnaXN0cnktdXNlciAvb3B0L2Rsdm0vb3Zm
LWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcp
CiAgICBSRUdJU1RSWV9QQVNTV09SRD0kKGdyZXAgcmVnaXN0cnktcGFzc3dkIC9vcHQv
ZGx2bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCku
Ki9cMS9wJykKICAgIGlmIFtbIC1uICIkUkVHSVNUUllfVVNFUk5BTUUiICYmICIC1uICIk
UkVHSVNUUllfUEFTU1dPUkQiIF1dOyB0aGVuCiAgICAgIGRvY2tlciBsb2dpbiAtdSAk
UkVHSVNUUllfVVNFUk5BTUUgLXAgJFJFR0lTVFJZX1BBU1NXT1JEICRSRUdJU1RSWV9V
UkkKICAgIGVsc2UKICAgICAgZWNobyAiV2FybmluZzogdGhlIHJlZ2lzdHJ5J3MgdXNl
cm5hbWUgYW5kIHBhc3N3b3JkIGFyZSBpbnZhbGlkLCBta2lwcGluZyBEb2NrZXIgbG9n
aW4uIgogICAgZmkKICAgIAogICAgZG9ja2VyIHJ1biAtZCAkUkVHSVNUUllfVVJBBB
VEgvbnZpZGlhL2s4cy9jdWRhLXNhbXBsZTp2ZWN0b3JhZGQtY3VkYTExLjcuMS11Ymk4
CgotIHBhdGg6IC9vcHQvZGx2bS91dGlscy5zaAogIHBlcm1pc3Npb25zOiAnMDc1NScK
ICBjb250ZW50OiB8CiAgICAjIS9iaW4vYmFzaAogICAgZXJyb3JffZXhpdCgpIHsKICAg
ICAgZWNobyAiRXJyb3I6ICQxIiA+JjIKICAgICAgdm10b29sc2QgLS1jbWQgImluZm8t
c2V0IGd1ZXN0aW5mby52bXNlcnZpY2UuYm9vdHN0cmFwLmNvbmRpdGlvbiBmYWxzZSwg
REExXb3JrbG9hZEZhaWx1cmUsICQxIgogICAgICBleGl0IDEKICAgICAgICAgIGVj
a19wcm90b2NvbCgpIHsKICAgICAgICAgbG9jYWwgcHJveHlfdXJsPSQxCiAgICAgICBj
aGVj
```

Tableau 3-1. Image de conteneur d'exemples CUDA (suite)

Composant	Description
	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

qui correspond au script suivant au format texte brut :

```
#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"
```

Tableau 3-1. Image de conteneur d'exemples CUDA (suite)

Composant	Description
	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

Tableau 3-1. Image de conteneur d'exemples CUDA (suite)

Composant	Description
	<pre> 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" }</pre> ■ Image sur une ligne. Codez-le au format base64 <pre>docker run -d nvcr.io/nvidia/k8s/cuda-sample:*ngc_image_tag*</pre>

Tableau 3-1. Image de conteneur d'exemples CUDA (suite)

Composant	Description
	Par exemple, pour vectoradd-cuda11.7.1-ubi8, fournissez le script suivant au format base64 : `ZG9ja2VyIHJ1biAtZCBudmNyLmlvL252aWRpYS9rOHMvY3VkYS1zYW1wbGU6dmVjdG9yYWRkLWN1ZGExMS43LjEtdWJpOA==` qui correspond au script suivant au format texte brut : `docker run -d nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8` ■ Entrez les propriétés d'installation du pilote invité vGPU, telles que `vgpu-license` et `nvidia-portal-api-key`. ■ Fournissez les valeurs des propriétés requises pour un environnement déconnecté si nécessaire. Reportez-vous à la section Propriétés OVF des VM à apprentissage profond.
Sortie	■ Journaux d'installation du pilote invité vGPU dans `/var/log/vgpu-install.log`. Pour vérifier que le pilote invité vGPU est installé et que la licence est allouée, exécutez la commande suivante : `nvidia-smi -q |grep -i license` ■ Journaux de script cloud-init dans `/var/log/dl.log`.

PyTorch

Vous pouvez utiliser une VM à apprentissage profond avec une bibliothèque PyTorch pour découvrir l'IA conversationnelle, le traitement automatique des langues (NLP, Natural language processing) et d'autres types de modèles d'IA sur une VM. Reportez-vous à la page PyTorch.

Une fois la VM à apprentissage profond lancée, elle démarre une instance de JupyterLab sur laquelle les modules PyTorch sont installés et configurés.

Tableau 3-2. Image de conteneur PyTorch

Composant	Description
Image de conteneur	`nvcr.io/nvidia/pytorch:ngc_image_tag` Par exemple : `nvcr.io/nvidia/pytorch:23.10-py3` Pour plus d'informations sur les images de conteneur PyTorch prises en charge pour les VM à apprentissage profond, reportez-vous à la section Notes de mise à jour de VMware Deep Learning VM.
Entrées requises	Pour déployer une charge de travail PyTorch, vous devez définir les propriétés OVF de la machine virtuelle à apprentissage profond de la manière suivante : ■ Utilisez l'une des propriétés suivantes spécifiques à l'image PyTorch. ■ Script cloud-init. Codez-le au format base64.

```
#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/
```

Tableau 3-2. Image de conteneur PyTorch (suite)

Composant	Description
	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 - 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

Tableau 3-2. Image de conteneur PyTorch (suite)

Composant	Description

```
      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"
    }
```

Par exemple, pour pytorch:23.10-py3, fournissez le script suivant au format base64 :

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCgog
ICAgREVGQVVMVF9SRUdfVVJJPSJudmNyLmlvIgogICAgUkVHSVNUUllfVVJJX1BBVEg9
JChncmVwIHJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4g
J3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCpCgogICAgaWYgW1sgLXogIiRS
RUdJU1RSWV9VUklfUEFUSCIgXV07IHRoZW4KICAgICAgIyBJZiBSRUdJU1RSWV9VUklf
UEFUSCBpcyBudWxsIG9yIGVtcHR5LCB1c2UgdGhlIGRlZmF1bHQgdmFsdWUKICAgICAg
UkVHSVNUUllfVVJJX1BBVEg9JERFRkFVTFRfUkVHX1VSSQogICAgICBlY2hvICJSRUdJ
U1RSWV9VUklfUEFUSCB3YXMgZW1wdHkuIFVzaW5nIGRlZmF1bHQ6ICRSRUdJU1RSWV9V
UklfUEFUSCIKICAgIGZpCiAgICAKICAgICMgSWYgUkVHSVNUUllfVVJJX1BBVEggY29u
dGFpbnMgJy8nLCBleHRyYWN0IHRoZSBVUkkgcGFydAogICAgaWYgW1sgJFJFR0lTVFJZ
X1VSSV9QQVRIID09ICoiLyIqIF1dOyB0aGVuCiAgICAgIFJFR0lTVFJZX1VSST0kKGVj
aG8gIiRSRUdJU1RSWV9VUklfUEFUSCIgfCBjdXQgLWQnLycgLWYxKQogICAgZWxzZQog
ICAgICBSRUdJU1RSWV9VUkk9JFJFR0lTVFJZX1VSSV9QQVRICiAgICBmaQogICAgICAg
UkVHSVNUUllfVVNFUk5BTUU9JChncmVwIHJlZ2lzdHJ5LXVzZXIgL29wdC9kbHZtL292
Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3An
KQogICAgUkVHSVNUUllfUEFTU1dPUkQ9JChncmVwIHJlZ2lzdHJ5LXBhc3N3ZCAvb3B0
L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwp
LiovXDEvcCpCiAgICBpZiBbbWyAtbiAiJFJFR0lTVFJZX1VTRVJOQU1FIiAmJiAtbiAi
JFJFR0lTVFJZX1BBU1NXT1JEIiBdXTsgdGhlbgogICAgICBkb2NrZXIgbG9naW4gLXUg
JFJFR0lTVFJZX1VTRVJOQU1FIC1wICRSRUdJU1RSWV9QQVNTV09SRCAkUkVHSVNUUllf
VVJJCiAgICBlbHNlCiAgICAgIGVjaG8gIldhcm5pbmc6IHRoZSByZWdpc3RyeSdzIHVz
ZXJuYW1lIGFuZCBwYXNzd29yZCBhcmUgaW52YWxpZCwgU2tpcHBpbmcgRG9ja2VyIGxv
Z2luLiIKICAgIGZpCgogICAgZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC1wIDg4ODg6
ODg4OCAkUkVHSVNUUllfVVJJX1BBVEgvbnZpZGlhL3B5dG9yY2g6MjMuMTAtcHkzIC91
c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9vdCAtLW1wPSogLS1wb3J0
PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAudG9rZW49JycgLS1Ob3RlYm9v
a0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGlyPS93b3Jrc3BhY2UKCi0g
```

Tableau 3-2. Image de conteneur PyTorch (suite)

Composant	Description
	(suite ci-dessous)


```
cGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNv
bnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAgICBl
Y2hvICJFcnJvcjogJDEiID4mMgogICAgICB2bXRvb2xzZCAtLWNtZCAiaW5mby1zZXQg
Z3Vlc3RpbmZvLnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIGZhbHNlLCBETFdv
cmtsb2FkRmFpbHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgfQoKICAgIGNoZWNrX3By
b3RvY29sKCkgewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQKICAg
ICAgbG9jYWwgc3VwcG9ydGVkX3Byb3RvY29scz0oIiRAIikKICAgICAgaWYgW1sgLW4g
IiR7cHJveHlfdXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm90b2NvbD0kKGVj
aG8gIiR7cHJveHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBwcmlu
dCAkMTsgZWxzZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2NvbCIg
XTsgdGhlbgogICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMgcHJvdG9jb2wgcHJvdmlk
ZWQuIFNraXBwaW5nIHByb3RvY29sIGNoZWNrLiIKICAgICAgICAgIHJldHVybiAwCiAg
ICAgICAgZmkKICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQogICAg
ICAgIGZvciB2YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwogICAg
ICAgICAgaWYgW1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9IiBdXTsgdGhlbgogICAg
ICAgICAgICBwcm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgIGJyZWFrCiAg
ICAgICAgICBmaQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2NvbF9p
bmNsdWRlZH0iID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0ICJV
bnN1cHBvcnRlZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1cHBvcnRlZCBwcm90b2Nv
bHMgYXJlOiAke3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgIGZpCiAgICAg
IGZpCiAgICB9CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xzCiAg
ICBzZXRfcHJveHkoKSB7CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIk
QCIpCgogICAgICBDT05GSUdfSlNPTl9CQVNFNjQ9JChncmVwICdjb25maWctanNvbicg
L29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14i
XSpcKS4qL1wxL3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdfSlNP
Tl9CQVNFNjR9IHwgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgSFRUUF9QUk9YWV9VUkw9
JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0dHBfcHJveHkgLy8gZW1w
dHknKQogICAgICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09OfSIg
fCBqcSAtciAnLmh0dHBzX3Byb3h5IC8vIGVtcHR5JykKICAgICAgaWYgW1sgJD8gLW5l
IDAgfHwgKC16ICIke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJPWFlf
VVJMfSIpIF1dOyB0aGVuCiAgICAgICAgZWNobyAiSW5mbzogVGhlIGNvbmZpZy1qc29u
IHdhcyBwYXJzZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kLiIKICAg
ICAgICByZXR1cm4gMAogICAgICBmaQoKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRU
UF9QUk9YWV9VUkx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAgY2hl
Y2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVkX3Byb3Rv
Y29sc1tAXX0iCgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldGMvZW52
aXJvbm1lbnQ7IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0ke0hU
VFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBTX1BS
T1hZX1VSTH0KICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJPWFlfVVJM
fQogICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH0KICAg
ICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0Yy9l
bnZpcm9ubWVudAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAgICAgIGZp
CiAgICAgIAogICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJveHkKICAg
ICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5kCiAg
ICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJPWFk9
JHtIVFRQX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BST1hZ
PSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25tZW50PVwiTk9fUFJPWFk9
bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9ja2Vy
LnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVsb2Fk
CiAgICAgIHN5c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAiSW5mbzog
ZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1cmVkIHRv
IHVzZSB0aGUgcHJveHkgc2V0dGluZ3MiCiAgICB9
```


qui correspond au script suivant au format texte brut.

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

Tableau 3-2. Image de conteneur PyTorch (suite)

Composant	Description
	#!/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 ""}')

Tableau 3-2. Image de conteneur PyTorch (suite)

Composant	Description
	<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>

Tableau 3-2. Image de conteneur PyTorch (suite)

Composant	Description
	<pre>echo "Info: docker and system environment are now configured to use the proxy settings" }</pre> ■ Image sur une ligne. Codez-le au format base64. <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> Par exemple, pour pytorch:23.10-py3, fournissez le script suivant au format base64 : <pre>ZG9ja2VyIHJ1biAtZCAtcCA4ODg4Ojg4ODggbnZjci5pby9udmlkaWEvcHl0b3JjaDoyMy4xMC1weTMgL3Vzci9sb2NhbC9iaW4vanVweXRlciBsYWIgLS1hbGxvdy1yb290IC0taXA9KiAtLXBvcnQ9ODg4OCAtLW5vLWJyb3dzZXIgLS1Ob3RlYm9va0FwcC50b2tlbj0nJyAtLU5vdGVib29rQXBwLmFsbG93X29yaWdpbj0nKicgLS1ub3RlYm9vay1kaXI9L3dvcmtzcGFjZQ==</pre> qui correspond au script suivant au format texte brut : <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> ■ Entrez les propriétés d'installation du pilote invité vGPU, telles que `vgpu-license` et `nvidia-portal-api-key`. ■ Fournissez les valeurs des propriétés requises pour un environnement déconnecté si nécessaire. Reportez-vous à la section Propriétés OVF des VM à apprentissage profond.
Sortie	■ Journaux d'installation du pilote invité vGPU dans `/var/log/vgpu-install.log`. Pour vérifier que le pilote invité vGPU est installé, exécutez la commande `nvidia-smi`. ■ Journaux de script cloud-init dans `/var/log/dl.log`. ■ Conteneur PyTorch. Pour vérifier que le conteneur PyTorch est en cours d'exécution, exécutez les commandes `sudo docker ps -a` et `sudo docker logs container_id`. ■ Instance de JupyterLab à laquelle vous pouvez accéder sur `http://dl_vm_ip:8888` Dans le terminal de JupyterLab, vérifiez que les fonctionnalités suivantes sont disponibles dans le bloc-notes : ■ Pour vérifier que JupyterLab peut accéder à la ressource vGPU, exécutez `nvidia-smi`. ■ Pour vérifier que les modules associés à PyTorch sont installés, exécutez `pip show`.

TensorFlow

Vous pouvez utiliser une VM à apprentissage profond avec une bibliothèque TensorFlow pour découvrir l'IA conversationnelle, le NLP et d'autres types de modèles d'IA sur une VM. Reportez-vous à la page TensorFlow.

Une fois la VM à apprentissage profond lancée, elle démarre une instance de JupyterLab sur laquelle les modules TensorFlow sont installés et configurés.

Tableau 3-3. Image de conteneur TensorFlow

Composant	Description
Image de conteneur	`nvcr.io/nvidia/tensorflow:ngc_image_tag` Par exemple : `nvcr.io/nvidia/tensorflow:23.10-tf2-py3` Pour plus d'informations sur les images de conteneur TensorFlow prises en charge pour les VM à apprentissage profond, reportez-vous à la section Notes de mise à jour de VMware Deep Learning VM.
Entrées requises	Pour déployer une charge de travail TensorFlow, vous devez définir les propriétés OVF de la machine virtuelle à apprentissage profond de la manière suivante : ■ Utilisez l'une des propriétés suivantes spécifiques à l'image TensorFlow. ■ Script cloud-init. Codez-le au format base64.

```
#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/
```

Tableau 3-3. Image de conteneur TensorFlow (suite)

Composant	Description
	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

Tableau 3-3. Image de conteneur TensorFlow (suite)

Composant	Description

```
      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"
    }
```

Par exemple, pour tensorflow:23.10-tf2-py3, fournissez le script suivant au format base64 :

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkb3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCiAg
ICAKICAgIERFRkFVTFRfUkVHX1VSST0ibnZjci5pbyIKICAgIFJFR0lTVFJZX1VSSV9Q
QVRIPSQoZ3JlcCByZWdpc3RyeS11cmkgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2Vk
IC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIGlmIFtbIC16
ICIkUkVHSVNUUllfVVJJX1BBVEgiIF1dOyB0aGVuCiAgICAgICMgSWYgUkVHSVNUUllf
VVJJX1BBVEggaXMgbnVsbCBvciBlbXB0eSwgdXNlIHRoZSBkZWZhdWx0IHZhbHVlCiAg
ICAgIFJFR0lTVFJZX1VSSV9QQVRIPSRERUZBVUxUX1JFR19VUkkKICAgICAgZWNobyAi
UkVHSVNUUllfVVJJX1BBVEggd2FzIGVtcHR5LiBVc2luZyBkZWZhdWx0OiAkUkVHSVNU
UllfVVJJX1BBVEgiCiAgICBmaQogICAgCiAgICAjIElmIFJFR0lTVFJZX1VSSV9QQVRI
IGNvbnRhaW5zICcvJywgZXh0cmFjdCB0aGUgVVJJIHBhcnQKICAgIGlmIFtbICRSRUdJ
U1RSWV9VUklfUEFUSCA9PSAqIi8iKiBdXTsgdGhlbgogICAgICBSRUdJU1RSWV9VUkk9
JChlY2hvICIkUkVHSVNUUllfVVJJX1BBVEgiIHwgY3V0IC1kJy8nIC1mMSkKICAgIGVs
c2UKICAgICAgUkVHSVNUUllfVVJJX1BBVEg9JFSQoZ3JlcCByZWdpc3RyeS11c2VyIC9vcHQvZGx2
bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9c
MS9wJykKICAgIFJFR0lTVFJZX1BBU1NXT1JEPSQoZ3JlcCByZWdpc3RyeS1wYXNzd2Qg
L29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14i
XSpcKS4qL1wxL3AnKQogICAgaWYgW1sgLW4gIiRSRUdJU1RSWV9VU0VSTkFNRSIgJiYg
LW4gIiRSRUdJU1RSWV9QQVNTV09SRCIgXV07IHRoZW4KICAgICAgZG9ja2VyIGxvZ2lu
IC11ICRSRUdJU1RSWV9VU0VSTkFNRSAtcCAkUkVHSVNUUllfUEFTU1dPUkQgJFJFR0lT
VFJZX1VSSQogICAgZWxzZQogICAgICBlY2hvICJXYXJuaW5nOiB0aGUgcmVnaXN0cnkn
cyB1c2VybmFtZSBhbmQgcGFzc3dvcmQgYXJlIGludmFsaWQsIFNraXBwaW5nIERvY2tl
ciBsb2dpbi4iCiAgICBmaQogICAgCiAgICAgICBkb2NrZXIgcnVuIC1kIC0tZ3B1cyBhbGwg
LXAgODg4ODo4ODg4ICRSRUdJU1RSWV9VUklfUEFUSC9udmlkaWEvdGVuc29yZmxvdzoy
My4xMC10ZjItcHkzIC91c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9v
dCAtLWlwPSogLS1wb3J0PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAudG9r
```

Tableau 3-3. Image de conteneur TensorFlow (suite)

Composant	Description


```
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
```


qui correspond au script suivant au format texte brut :

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

Tableau 3-3. Image de conteneur TensorFlow (suite)

Composant	Description

```
  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
```

Tableau 3-3. Image de conteneur TensorFlow (suite)

Composant	Description

```
(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
```

Tableau 3-3. Image de conteneur TensorFlow (suite)

Composant	Description
	<pre>echo "Info: docker and system environment are now configured to use the proxy settings" }</pre> ■ Image sur une ligne. Codez-le au format base64. <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> Par exemple, pour tensorflow:23.10-tf2-py3, fournissez le script suivant au format base64 : <pre>ZG9ja2VyIHJ1biAtZCAtcCA4ODg4Ojg4ODggbnZjci5pby9udmlkaWEvdGVuc29yZmxvdzoyMy4xMC10ZjItcHkzIC91c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9vdCAtLWlwPSogLS1wb3J0PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAudG9rZW49JycgLS1Ob3RlYm9va0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGlyPS93b3Jrc3BhY2U=</pre> qui correspond au script suivant au format texte brut : <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> ■ Entrez les propriétés d'installation du pilote invité vGPU, telles que `vgpu-license` et `nvidia-portal-api-key`. ■ Fournissez les valeurs des propriétés requises pour un environnement déconnecté si nécessaire. Reportez-vous à la section Propriétés OVF des VM à apprentissage profond.
Sortie	■ Journaux d'installation du pilote invité vGPU dans `/var/log/vgpu-install.log`. Pour vérifier que le pilote invité vGPU est installé, connectez-vous à la VM via SSH et exécutez la commande `nvidia-smi`. ■ Journaux de script cloud-init dans `/var/log/dl.log`. ■ Conteneur TensorFlow. Pour vérifier que le conteneur TensorFlow est en cours d'exécution, exécutez les commandes `sudo docker ps -a` et `sudo docker logs container_id`. ■ Instance de JupyterLab à laquelle vous pouvez accéder sur `http://dl_vm_ip:8888`. Dans le terminal de JupyterLab, vérifiez que les fonctionnalités suivantes sont disponibles dans le bloc-notes : ■ Pour vérifier que JupyterLab peut accéder à la ressource vGPU, exécutez `nvidia-smi`. ■ Pour vérifier que les modules associés à TensorFlow sont installés, exécutez `pip show`.

Exportateur DCGM

Vous pouvez utiliser une VM à apprentissage profond avec l'exportateur Data Center GPU Manager (DCGM) pour surveiller la santé des GPU et obtenir des mesures de ceux-ci qui sont utilisés par une charge de travail DL, à l'aide de NVIDIA DCGM, Prometheus et Grafana.

Reportez-vous à la page Exportateur DCGM.

Dans une VM à apprentissage profond, exécutez le conteneur de l'exportateur DCGM avec une charge de travail DL qui effectue des opérations d'IA. Une fois la VM à apprentissage profond démarrée, l'exportateur DCGM est prêt à collecter des mesures de vGPU et à exporter les données vers une autre application pour une surveillance et une visualisation accrues. Vous pouvez exécuter la charge de travail DL surveillée dans le cadre du processus cloud-init ou à partir de la ligne de commande après le démarrage de la machine virtuelle.

Tableau 3-4. Image de conteneur de l'exportateur DCGM

Composant	Description
Image de conteneur	`nvcr.io/nvidia/k8s/dcgm-exporter:ngc_image_tag` Par exemple : `nvcr.io/nvidia/k8s/dcgm-exporter:3.2.5-3.1.8-ubuntu22.04` Pour plus d'informations sur les images de conteneur de l'exportateur DCGM qui sont prises en charge pour les VM à apprentissage profond, reportez-vous à la section Notes de mise à jour de VMware Deep Learning VM.
Entrées requises	Pour déployer une charge de travail de l'exportateur DCGM, vous devez définir les propriétés OVF de la machine virtuelle à apprentissage profond de la manière suivante : ■ Utilisez l'une des propriétés suivantes propres à l'image de l'exportateur DCGM. ■ Script cloud-init. Codez-le au format base64.

```
#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
```

Tableau 3-4. Image de conteneur de l'exportateur DCGM (suite)

Composant	Description
	`$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}" "$`

Tableau 3-4. Image de conteneur de l'exportateur DCGM (suite)

Composant	Description

```
{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"
    }
```

Par exemple, pour une VM à apprentissage profond avec une instance de l'exportateur DCGM dcgm-exporter:3.2.5-3.1.8-ubuntu22.04 préinstallée, fournissez le script suivant au format base64

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCiAg
ICAKICAgIERFRkFVTFRfUkVHX1VSST0ibnZjci5pbyIKICAgIFJFR0lTVFJZX1VSSV9Q
QVRIPSQoZ3JlcCByZWdpc3RyeS11cmkgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2Vk
IC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIGlmIFtbIC16
ICIkUkVHSVNUUllfVVJJX1BBVEgiIF1dOyB0aGVuCiAgICAgICMgSWYgUkVHSVNUUllf
VVJJX1BBVEggaXMgbnVsbCBvciBlbXB0eSwgdXNlIHRoZSBkZWZhdWx0IHZhbHVlCiAg
ICAgIFJFR0lTVFJZX1VSSV9QQVRIPSRERUZBVUxUX1JFR19VUkkKICAgICAgZWNobyAi
UkVHSVNUUllfVVJJX1BBVEggd2FzIGVtcHR5LiBVc2luZyBkZWZhdWx0OiAkUkVHSVNU
UllfVVJJX1BBVEgiCiAgICBmaQogICAgICAjIElmIFJFR0lTVFJZX1VSSV9QQVRI
IGNvbnRhaW5zICcvJywgZXh0cmFjdCB0aGUgVVJJIHBhcnQKICAgIGlmIFtbICRSRUdJ
U1RSWV9VUklfUEFUSCA9PSAqIi8iKiBdXTsgdGhlbgogICAgICBSRUdJU1RSWV9VUkk9
JChlY2hvICIkUkVHSVNUUllfVVJJX1BBVEgiIHwgY3V0IC1kJy8nIC1mMSkKICAgIGVs
c2UKICAgICAgUkVHSVNUUllfVVJJX1BBVEg9IiIKICAgIGZpCiAgICBlY2hvICJSRUdJ
U1RSWV9VUklfUEFUSD0kUkVHSVNUUllfVVJJX1BBVEgiCiAgICBlY2hvICJSRUdJU1RS
WV9VUklfVFlQRT0kUkVHSVNUUllfVFlQRSIKICAgIGVjaG8gIlJFR0lTVFJZX1RSVVNU
RURfQ0VSVFM9JChlY2hvICIkUkVHSVNUUllfVVJJX1BBVEgiIHwgY3V0IC1kJy8nIC1m
MSkKICAgIGVsc2UKICAgICAgUkVHSVNUUllfVVJJX1BBVEg9IiIKICAgIGZpCiAgICBl
cmFnZQoKICAgIGlmIFtbIC16ICIkUkVHSVNUUllfVVJJX1BBVEgiIF1dOyB0aGVuCiAg
ICAgIGVjaG8gIlJFR0lTVFJZX1RSVVNURURfQ0VSVFM9IiIKICAgICAgZWNobyAiUkVH
SVNUUllfVVJJX1BBVEg9IiIKICAgIGZpCiAgICBlY2hvICJSRUdJU1RSWV9VUklfUEFU
SCA9PSAqIi8iKiBdXTsgdGhlbgogICAgICBSRUdJU1RSWV9VUklfUEFUSD0kUkVHSVNU
UllfVVJJX1BBVEgiCiAgICBmaQogICAgZWNobyAiUkVHSVNUUllfVVJJX1BBVEg9IiRS
RUdJU1RSWV9VUklfUEFUSCIKICAgIGVjaG8gIlJFR0lTVFJZX1RSVVNURURfQ0VSVFM9
IiRSRUdJU1RSWV9UUlVTVF9SRCIgXV07IHRoZW4KICAgICAgZG9ja2VyIGxvZ2luIC11
ICRSRUdJU1RSWV9VU0VSTkFNRSAtcCAkUkVHSVNUUllfUEFTU1dPUkQgJFJFR0lT
VFJZX1VSSQogICAgZWxzZQogICAgICBlY2hvICJXYXJuaW5nOiB0aGUgcmVnaXN0cnkn
cyB1c2VybmFtZSBhbmQgcGFzc3dvcmQgYXJlIGludmFsaWQsIFNraXBwaW5nIERvY2tl
ciBsb2dpbi4iCiAgICBmaQoKICAgIGRvY2tlciBydW4gLWQgLS1ncHVzIGFsbCAtLWNh
cC1hZGQgU1lTX0FETUlOIC0tcm0gLXAgOTQwMDo5NDAwICRSRUdJU1RSWV9VUklfUEFU
SC9udmlkaWEvazhzL2RjZ20tZXhwb3J0ZXI6My4yLjUtMy4xLjgtdWJ1bnR1MjIuMDQK
Ci0gcGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1Jwog
IGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAg
```

Tableau 3-4. Image de conteneur de l'exportateur DCGM (suite)

Composant	Description


```
ICBlY2hvICJFcnJvcjogJDEiID4mMgogICAgICB2bXRvb2xzZCAtLWNtZCAiaW5mby1z
ZXQgZ3Vlc3RpbmZvLnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIGZhbHNlLCBE
TFdvcmtsb2FkRmFpbHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgfQoKICAgIGNoZWNr
X3Byb3RvY29sKCkgewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQK
ICAgICAgbG9jYWwgc3VwcG9ydGVkX3Byb3RvY29scz0oIiRAIikKICAgICAgaWYgW1sg
LW4gIiR7cHJveHlfdXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm90b2NvbD0k
KGVjaG8gIiR7cHJveHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBw
cmludCAkMTsgZWxzZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2Nv
bCIgXTsgdGhlbgogICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMgcHJvdG9jb2wgcHJv
dmlkZWQuIFNraXBwaW5nIHByb3RvY29sIGNoZWNrLiIKICAgICAgICAgIHJldHVybiAw
CiAgICAgICAgZmkKICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQog
ICAgICAgIGZvciB2YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwog
ICAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9IiBdXTsgdGhlbgog
ICAgICAgICAgICBwcm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgIGJyZWFr
CiAgICAgICAgICBmaQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2Nv
bF9pbmNsdWRlZH0iID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0
ICJVbnN1cHBvcnRlZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1cHBvcnRlZCBwcm90
b2NvbHMgYXJlOiAke3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgIGZpCiAg
ICAgIGZpCiAgICB9CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xz
CiAgICBzZXRfcHJveHkoKSB7CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9
KCIkQCIpCgogICAgICBDT05GSUdfSlNPTl9CQVNFNjQ9JChncmVwICdjb25maWctanNv
bicgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9Ilwo
W14iXSpcKS4qL1wxL3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdf
SlNPTl9CQVNFNjR9IHwgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgSFRUUF9QUk9YWV9V
Ukw9JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0dHBfcHJveHkgLy8g
ZW1wdHknKQogICAgICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09O
fSIgfCBqcSAtciAnLmh0dHBzX3Byb3h5IC8vIGVtcHR5JykKICAgICAgaWYgW1sgJD8g
LW5lIDAgfHwgKC16ICIke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJP
WFlfVVJMfSIpIF1dOyB0aGVuCiAgICAgICAgZWNobyAiSW5mbzogVGhlIGNvbmZpZy1q
c29uIHdhcyBwYXJzZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kLiIK
ICAgICAgICByZXR1cm4gMAogICAgICBmaQoKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7
SFRUUF9QUk9YWV9VUkx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAg
Y2hlY2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVkX3By
b3RvY29sc1tAXX0iCgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldGMv
ZW52aXJvbm1lbnQ7IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0k
e0hUVFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBT
X1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJPWFlf
VVJMfQogICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH0K
ICAgICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0
Yy9lbnZpcm9ubWVudAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAgICAg
IGZpCiAgICAgICAgIAogICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJveHkK
ICAgICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5k
CiAgICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJP
WFk9JHtIVFRQX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BS
T1hZPSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25tZW50PVwiTk9fUFJP
WFk9bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9j
a2VyLnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVs
b2FkCiAgICAgIHN5c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAiSW5m
bzogZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1cmVk
IHRvIHVzZSB0aGUgcHJveHkgc2V0dGluZ3MiCiAgICB9
```


qui correspond au script suivant au format texte brut :

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

Tableau 3-4. Image de conteneur de l'exportateur DCGM (suite)

Composant	Description
	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."

Tableau 3-4. Image de conteneur de l'exportateur DCGM (suite)

Composant	Description

```
          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
```

Tableau 3-4. Image de conteneur de l'exportateur DCGM (suite)

Composant	Description
	echo "Info: docker and system environment are now configured to use the proxy settings" } **Note** Vous pouvez également ajouter au script cloud-init les instructions d'exécution de la charge de travail DL dont vous souhaitez mesurer les performances de GPU avec l'exportateur DCGM. ■ Image sur une ligne. Codez-le au format base64. `docker run -d --gpus all --cap-add SYS_ADMIN --rm -p 9400:9400 nvcr.io/nvidia/k8s/dcgm-exporter:`*`ngc_image_tag`*`-ubuntu22.04` Par exemple, pour dcgm-exporter:3.2.5-3.1.8-ubuntu22.04, fournissez le script suivant au format base64 : `ZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tY2FwLWFkZCBTWVNfQURNSU4gLS1ybSAtcCA5NDAwOjk0MDAgbnZjci5pby9udmlkaWEvazhzL2RjZ20tZXhwb3J0ZXI6My4yLjUtMy4xLjgtdWJ1bnR1MjIuMDQ=` qui correspond au script suivant au format texte brut : `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` ■ Entrez les propriétés d'installation du pilote invité vGPU, telles que `vgpu-license` et `nvidia-portal-api-key`. ■ Fournissez les valeurs des propriétés requises pour un environnement déconnecté si nécessaire. Reportez-vous à la section Propriétés OVF des VM à apprentissage profond.
Sortie	■ Journaux d'installation du pilote invité vGPU dans `/var/log/vgpu-install.log`. Pour vérifier que le pilote invité vGPU est installé, connectez-vous à la VM via SSH et exécutez la commande `nvidia-smi`. ■ Journaux de script cloud-init dans `/var/log/dl.log`. ■ Exportateur DCGM auquel vous pouvez accéder à l'adresse `http://`*`dl_vm_ip`*`:9400`. Ensuite, dans la VM à apprentissage profond, exécutez une charge de travail DL et visualisez les données sur une autre machine virtuelle à l'aide de Prometheus à l'adresse `http://`*`visualization_vm_ip`*`:9090` et Grafana à l'adresse `http://`*`visualization_vm_ip`*`:3000`.

Exécuter une charge de travail DL sur la VM à apprentissage profond

Exécutez la charge de travail DL pour laquelle vous souhaitez collecter des mesures vGPU et exportez les données vers une autre application pour une surveillance et une visualisation accrues.

1 Connectez-vous à la VM à apprentissage profond en tant que **vmware** via SSH.

2 Ajoutez le compte d'utilisateur **vmware** au groupe **docker** en exécutant la commande suivante.

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

3 Exécutez le conteneur pour la charge de travail DL, en l'extrayant du catalogue NVIDIA NGC ou d'un registre de conteneur local.

Par exemple, pour exécuter la commande suivante afin de lancer l'image tensorflow:23.10-tf2-py3 à partir de NVIDIA NGC :

```
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 Utilisez la charge de travail DL pour le développement de l'IA.

Installer Prometheus et Grafana

Vous pouvez visualiser et surveiller les mesures vGPU à partir de la machine virtuelle de l'exportateur DCGM sur une machine virtuelle exécutant Prometheus et Grafana.

1 Créez une VM de visualisation sur laquelle Docker Community Engine est installé.

2 Connectez-vous à la VM via SSH et créez un fichier YAML pour 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 Créez un chemin de données.

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

4 Créez un fichier Docker Compose pour installer Prometheus et Grafana.

```
$ 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 Démarrez les conteneurs Prometheus et Grafana.

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

Afficher les mesures vGPU dans Prometheus

Vous pouvez accéder à Prometheus à l'adresse `http://visualization-vm-ip:9090`. Vous pouvez afficher les informations de vGPU suivantes dans l'interface utilisateur de Prometheus :

Informations	Section de l'interface utilisateur
Mesures de vGPU brutes de la VM à apprentissage profond	**État > Cible** Pour afficher les mesures vGPU brutes de la VM à apprentissage profond, cliquez sur l'entrée du point de terminaison.
Expressions graphiques	1 Dans la barre de navigation principale, cliquez sur l'onglet **Graphique**. 2 Entrez une expression et cliquez sur **Exécuter**.

Pour plus d'informations sur l'utilisation de Prometheus, reportez-vous à la documentation de Prometheus.

Visualiser les mesures dans Grafana

Définissez Prometheus comme source de données pour Grafana et visualisez les mesures vGPU à partir de la VM à apprentissage profond dans un tableau de bord.

1 Accédez à Grafana à l'adresse `http://visualization-vm-ip:3000` en utilisant par défaut le nom d'utilisateur **admin** et le mot de passe `admin`.

2 Ajoutez Prometheus comme première source de données, en vous connectant à *visualization-vm-ip* sur le port 9090.

3 Créez un tableau de bord avec les mesures vGPU.

Pour plus d'informations sur la configuration d'un tableau de bord à l'aide d'une source de données Prometheus, reportez-vous à la documentation de Grafana.

Serveur d'inférence Triton

Vous pouvez utiliser une VM à apprentissage profond avec un serveur d'inférence Triton pour charger un référentiel de modèles et recevoir des demandes d'inférence.

Reportez-vous à la page Serveur d'inférence Triton.

Tableau 3-5. Image de conteneur du serveur d'inférence Triton

Composant	Description
Image de conteneur	`nvcr.io/nvidia/tritonserver:ngc_image_tag` Par exemple : `nvcr.io/nvidia/tritonserver:23.10-py3` Pour plus d'informations sur les images de conteneur du serveur d'inférence Triton prises en charge pour les VM à apprentissage profond, reportez-vous à la section Notes de mise à jour de VMware Deep Learning VM.
Entrées requises	Pour déployer une charge de travail du serveur d'inférence Triton, vous devez définir les propriétés OVF de la machine virtuelle à apprentissage profond de la manière suivante : ■ Utilisez l'une des propriétés suivantes qui sont spécifiques à l'image du serveur d'inférence Triton. ■ Script cloud-init. Codez-le au format base64. `#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`

Tableau 3-5. Image de conteneur du serveur d'inférence Triton (suite)

Composant	Description
	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:*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

Tableau 3-5. Image de conteneur du serveur d'inférence Triton (suite)

Composant	Description

```
      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"
    }
```

Par exemple, pour tritonserver:23.10-py3, fournissez le script suivant au format base64 :

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkb3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCgog
ICAgREVGQVVMVF9SRUdfVVJJPSJudmNyLmlvIgogICAgUkVHSVNUUllfVVJJX1BBVEg9
JChncmVwIHJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4g
J3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCpCgogICAgaWYgW1sgLXogIiRS
RUdJU1RSWV9VUklfUEFUSCIgXV07IHRoZW4KICAgICAgIyBJZiBSRUdJU1RSWV9VUklf
UEFUSCBpcyBudWxsIG9yIGVtcHR5LCB1c2UgdGhlIGRlZmF1bHQgdmFsdWUKICAgICAg
UkVHSVNUUllfVVJJX1BBVEg9JERFRkFVTFRfUkVHX1VSSQogICAgICBlY2hvICJSRUdJ
U1RSWV9VUklfUEFUSCB3YXMgZW1wdHkuIFVzaW5nIGRlZmF1bHQ6ICRSRUdJU1RSWV9V
UklfUEFUSCIKICAgIGZpCiAgICAKICAgICMgSWYgUkVHSVNUUllfVVJJX1BBVEggY29u
dGFpbnMgJy8nLCBleHRyYWN0IHRoZSBVUkkgcGFydAogICAgaWYgW1sgJFJFR0lTVFJZ
X1VSSV9QQVRIID09ICoiLyIqIF1dOyB0aGVuCiAgICAgIFJFR0lTVFJZX1VSST0kKGVj
aG8gIiRSRUdJU1RSWV9VUklfUEFUSCIgfCBjdXQgLWQnLycgLWYxKQogICAgZWxzZQog
ICAgICBSRUdJU1RSWV9VUkk9JFJFR0lTVFJZX1VSSV9QQVRICiAgICBmaQogICAgICAo
UkVHSVNUUllfVVNFUk5BTUU9JChncmVwIHJlZ2lzdHJ5LXVzZXIgL29wdC9kbHZtL292
Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3An
KQogICAgUkVHSVNUUllfUEFTU1dPUkQ9JChncmVwIHJlZ2lzdHJ5LXBhc3N3ZCAvb3B0
L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwp
LiovXDEvcCpCiAgICBpZiBbWyAtbiAiJFJFR0lTVFJZX1VTRVJOQU1FIiAmJiAtbiAi
JFJFR0lTVFJZX1BBU1NXT1JEIiBdXTsgdGhlbgogICAgICBkb2NrZXIgbG9naW4gLXUg
JFJFR0lTVFJZX1VTRVJOQU1FIC1wICRSRUdJU1RSWV9QQVNTV09SRCAkUkVHSVNUUllf
VVJJCiAgICBlbHNlCiAgICAgIGVjaG8gIldhcm5pbmc6IHRoZSByZWdpc3RyeSdzIHVz
ZXJuYW1lIGFuZCBwYXNzd29yZCBhcmUgaW52YWxpZCwgU2tpcHBpbmcgRG9ja2VyIGxv
Z2luLiIKICAgIGZpCgogICAgZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tcm0gLXAg
ODAwMDo4MDAwIC1wIDgwMDE6ODAwMSAtcCA4MDAyOjgwMDIgLXYgL2hvbWUvdm13YXJl
L21vZGVsX3JlcG9zaXRvcnk6L21vZGVscycyAkUkVHSVNUUllfVVJJX1BBVEgvbnZpZGlh
L3RyaXRvbnNlcnZlcjoyMy4xMC1weTMgdHJpdG9uc2VydmVyIC0tbW9kZWwtcmVwb3Np
```

Tableau 3-5. Image de conteneur du serveur d'inférence Triton (suite)

Composant	Description
	dG9yeT0vbW9kZWxzIC0tbW9kZWwtY29udHJvbC1tb2RlPXBvbGwKCi0gcGF0aDogL29w dC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwK ICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAgICBlY2hvICJFcnJv cjogJDEiID4mMgogICAgICB2bXRvb2xzZCAtLWNtZCAiaW5mby1zZXQgZ3Vlc3RpbmZv LnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIGZhbHNlLCBETFdvcmtsb2FkRmFp bHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgfQoKICAgIGNoZWNrX3Byb3RvY29sKCkg ewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQKICAgICAgbG9jYWwg c3VwcG9ydGVkX3Byb3RvY29scz0oIiRAIikKICAgICAgaWYgW1sgLW4gIiR7cHJveHlf dXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm90b2NvbD0kKGVjaG8gIiR7cHJv eHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBwcmludCAkMTsgZWxz ZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2NvbCIgXTsgdGhlbgog ICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMgcHJvdG9jb2wgcHJvdmlkZWQuIFNraXBw aW5nIHByb3RvY29sIGNoZWNrLiKICAgICAgICAgIHJldHVybiAwCiAgICAgICAgZmkK ICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQogICAgICAgIGZvciB2 YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwogICAgICAgICAgaWYg W1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9IiBdXTsgdGhlbgogICAgICAgICAgICBw cm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgIGJyZWFrCiAgICAgICAgICAg aQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2NvbF9pbmNsdWRlZH0i ID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0ICJVbnN1cHBvcnRl ZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1cHBvcnRlZCBwcm90b2NvbHMgYXJlOiAk e3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgICAgZXQgLW4gIiR7cHJvdG9j CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xzCiAgICBBzZXRfcHJv eHkoKSB7CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIpCgogICAg ICBDT05GSUdfSlNPTl9CQVNFNjQ9JChncmVwICdjb25maWctanNvbicgL29wdC9kbHZt L292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wx L3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdfSlNPTl9CQVNFNjR9 IHwgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgSFRUUF9QUk9YWV9VUkw9JChlY2hvICIk e0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0dHBfcHJveHkgLy8gZW1wdHknKQogICAg ICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqcSAtciAn Lmh0dHBzX3Byb3h5IC8vIGVtcHR5JykKICAgICAgaWYgW1sgJD8gLW5lIDAgfHwgKC16 ICIke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJPWFlfVVJMfSIpIF1d OyB0aGVuCiAgICAgICAgZWNobyAiSW5mbzogVGhlIGNvbmZpZy1qc29uIHdhcyBwYXJz ZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kLiIKICAgICAgICAgcm4gMAogICAgICBmaQoKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUF9QUk9YWV9V Ukx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAgY2hlY2tfcHJvdG9j b2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0i CgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldGMvZW52aXJvbm1lbnQ7 IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0ke0hUVFBfUFJPWFlf VVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBTX1BST1hZX1VSTH0K ICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJPWFlfVVJMfQogICAgICAg IGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH0KICAgICAgICBleHBv cnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0Yy9lbnZpcm9ubWVu dAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAgICAgICAgZpCiAgICAgIAog ICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJveHkKICAgICAgbWtaXIg LXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5kCiAgICAgICAgIGVjaG8g IltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJPWFk9JHtIVFRQX1BS T1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BST1hZPSR7SFRUUFNf UFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25tZW50PVwiTk9fUFJPWFk9bG9jYWxob3N0 LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9ja2VyLnNlcnZpY2Uu ZC9wcm94eS5jb25mCiAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVsb2FkCiAgICAgICAg c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAiSW5mbzogZG9ja2VyIGFu ZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1cmVkIHRvIHVzZSB0aGUg cHJveHkgc2V0dGluZ3MiCiAgICB9

qui correspond au script suivant au format texte brut :

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

Tableau 3-5. Image de conteneur du serveur d'inférence Triton (suite)

Composant	Description

```
  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
```

Tableau 3-5. Image de conteneur du serveur d'inférence Triton (suite)

Composant	Description

```
(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
```

Tableau 3-5. Image de conteneur du serveur d'inférence Triton (suite)

Composant	Description
	<pre> echo "Info: docker and system environment are now configured to use the proxy settings" }</pre> ■ Image codée sur une ligne au format base64 <pre>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</pre> Par exemple, pour tritonserver:23.10-py3, fournissez le script suivant au format base64 : <pre>ZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tcm0gLXA4MDAwOjgwMDAgLXA4MDAxOjgwMDEgLXA4MDAyOjgwMDIgLXYgL2hvbWUvdm13YXJlL21vZGVsX3JlcG9zaXRvcnk6L21vZGVscyBudmNyLmlvL252aWRpYS90cml0b25zZXJ2ZXI6MjMuMTAtcHkzIHRyaXRvbnNlcnZlciAtLW1vZGVsLXJlcG9zaXRvcnk9L21vZGVscyAtLW1vZGVsLWNvbnRyb2wtbW9kZT1wb2xs</pre> qui correspond au script suivant au format texte brut : <pre>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</pre> ■ Entrez les propriétés d'installation du pilote invité vGPU, telles que `vgpu-license` et `nvidia-portal-api-key`. ■ Fournissez les valeurs des propriétés requises pour un environnement déconnecté si nécessaire. Reportez-vous à la section Propriétés OVF des VM à apprentissage profond.
Sortie	■ Journaux d'installation du pilote invité vGPU dans `/var/log/vgpu-install.log`. Pour vérifier que le pilote invité vGPU est installé, connectez-vous à la VM via SSH et exécutez la commande `nvidia-smi`. ■ Journaux de script cloud-init dans `/var/log/dl.log`. ■ Conteneur du serveur d'inférence Triton. Pour vérifier que le conteneur du serveur d'inférence Triton est en cours d'exécution, exécutez les commandes `sudo docker ps -a` et `sudo docker logs container_id`. Le référentiel de modèles du serveur d'inférence Triton se trouve dans `/home/vmware/model_repository`. Initialement, le référentiel de modèles est vide et le journal initial de l'instance du serveur d'inférence Triton indique qu'aucun modèle n'est chargé.

Créer un référentiel de modèles

Pour charger votre modèle de l'inférence, procédez comme suit :

1 Créez le référentiel de votre modèle.

Reportez-vous à la documentation du référentiel de modèles du serveur d'inférence Triton de NVIDIA.

2 Copiez le référentiel de modèles dans `/home/vmware/model_repository` afin que le serveur d'inférence Triton puisse le charger.

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

Envoyer des demandes d'inférence de modèles

1 Vérifiez que le serveur d'inférence Triton est sain et que les modèles sont prêts en exécutant cette commande dans la console de VM à apprentissage profond.

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

2 Envoyez une demande au modèle en exécutant cette commande sur la VM à apprentissage profond.

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

Pour plus d'informations sur l'utilisation du serveur d'inférence Triton, reportez-vous à la documentation du référentiel de modèles du serveur d'inférence Triton de NVIDIA.

NVIDIA RAG

Vous pouvez utiliser une VM à apprentissage profond pour créer des solutions de génération augmentée de récupération (RAG) avec un modèle Llama2.

Reportez-vous à la documentation Outil Docker Compose des applications NVIDIA RAG (nécessite des autorisations de compte spécifiques).

Tableau 3-6. Image du conteneur NVIDIA RAG

Composant	Description
Images et modèles de conteneur	`rag-app-text-chatbot.yaml` dans l'exemple de pipeline NVIDIA RAG. Pour plus d'informations sur les applications de conteneur NVIDIA RAG prises en charge pour les VM à apprentissage profond, reportez-vous à la section Notes de mise à jour de VMware Deep Learning VM.
Entrées requises	Pour déployer une charge de travail NVIDIA RAG, vous devez définir les propriétés OVF de la machine virtuelle à apprentissage profond de la manière suivante : ■ Entrez un script cloud-init. Codez-le au format base64. Par exemple, pour la version 24.03 de NVIDIA RAG, fournissez le script suivant : I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBwLnN oCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCi AgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0cmFwICdlc nJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3JrbG9hZCInIEVS UgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiCgogICAgY2F0IDw8RU9GID4gL29wdC9 kbHZtL2NvbmZpZy5qc29uCiAgICB7CiAgICAgICJfY29tbWVudCI6ICJUaGlzIHByb3ZpZG VzIGRlZmF1bHQgc3VwcG9ydCBmb3IgUkFHOiBUZW5zb3JSVCBpbmZlcmVuY2UsIGxsYW1hM i0xM2IgbW9kZWwsIGFuZCBIMTAweDIgR1BVIiwKICAgICAgInJhZyI6IHsKICAgICAgICAi b3JnX25hbWUiOiAiY29jZndnYThqcTJjIiwKICAgICAgICAib3JnX3RlYW1fbmFtZSI6ICJ uby10ZWFtIiwKICAgICAgICAicmFnX3JlcG9fbmFtZSI6ICJudmlkaWEvcGFpIiwCiAgIC AgICAgImxsbV9yZXBvX25hbWUiOiAibnZpZGlhL25pbSIsCiAgICAgICAgImVtYmVkX3Jlc G9fbmFtZSI6ICJudmlkaWEvbmVtby1yZXRyaWV2ZXIiLAogICAgICAgICJyYWdfbmFtZSI6 ICJyYWctZG9ja2VyLWNvbXBvc2UiLAogICAgICAgICJyYWdfdmVyc2lvbiI6ICIyNC4wMyI sCiAgICAgICAgImVtYmVkX25hbWUiOiAibnYtZW1iZWQtcWEiLAogICAgICAgICJlbWJlZF 90eXBlIjogIk5WLUVtYmVkLVFBIiwKICAgICAgICAiZW1iZWRfdmVyc2lvbiI6ICI0IiwKI CAgICAgICAiaW5mZXJlbmNlX3R5cGUiOiAidHJ0IiwKICAgICAgICAibGxtX25hbWUiOiAi bGxhbWEyLTEzYi1jaGF0IiwKICAgICAgICAibGxtX3ZlcnNpb24iOiAiaDEwMHgyX2ZwMTZ fMjQuMDIiLAogICAgICAgICJudW1fZ3B1IjogIjIiLAogICAgICAgICJoZl90b2tlbiI6IC JodWdnaW5nZmFjZSB0b2tlbiB0byBwdWxsIGxsbSBtb2RlbCwgdXBkYXRlIHdoZW4gdXNpb mcgdmxsbSBpbmZlcmVuY2UiLAogICAgICAgICJoZl9yZXBvIjogImh1Z2dpbmdmYWNlIGxs bSBtb2RlbCByZXBvc2l0b3J5LCB1cGRhdGUgd2hlbiB1c2luZyB2bGxtIGluZmVyZW5jZSI KICAgICAgfQogICAgfQogICAgRU9GCiAgICBDT05GSUdfSlNPTj0kKGNhdCAiL29wdC9kbH ZtL2NvbmZpZy5qc29uIikKICAgIElORkVSRU5DRV9UWVBFPSQoZWNobyAiJHtDT05GSUdfS lNPTn0iIHwganEgLXIgJy5yYWcuaW5mZXJlbmNlX3R5cGUnKQogICAgaWYgWyAiJHtJTkZF UkVOQ0VfVFlQRX0iID0gInRydCIgXTsgdGhlbgogICAgICByZXF1aXJlZF92YXJzPSgiT1J HX05BTUUiICJPUkdfVEVBTV9OQU1FIiAiUkFHX1JFUE9fTkFNRSIgIkxMTV9SRVBPX05BTU UiICJFTUJFRF9SRVBPX05BTUUiICJSQUdfTkFNRSIgIlJBR19WRVJTSU9OIiAiRU1CRURfT kFNRSIgIkVNQkVEX1RZUEUiICJFTUJFRF9WRVJTSU9OIiAiTExNX05BTUUiICJMTE1fVkVS U0lPTiIgIk5VTV9HUFUiKQogICAgZWxpZiBbICIke0lORkVSRU5DRV9UWVBFfSIgPSAidmx sbSIgXTsgdGhlbgogICAgICByZXF1aXJlZF92YXJzPSgiT1JHX05BTUUiICJPUkdfVEVBTV 9OQU1FIiAiUkFHX1JFUE9fTkFNRSIgIkxMTV9SRVBPX05BTUUiICJFTUJFRF9SRVBPX05BT UUiICJSQUdfTkFNRSIgIlJBR19WRVJTSU9OIiAiRU1CRURfTkFNRSIgIkVNQkVEX1RZUEUi ICJFTUJFRF9WRVJTSU9OIiAiTExNX05BTUUiICJOVU1fR1BVIiAiSEZfVE9LRU4iICJIR19 SRVBPIikKICAgIGVsc2UKICAgICAgZXJyb3JfZXhpdCAiSW5mZXJlbmNlIHR5cGUgJyR7SU 5GRVJFTkNFX1RZUEV9JyBpcyBub3QgcmVjb2duaXplZC4gTm8gYWN0aW9uIHdpbGwgYmUgd GFrZW4uIgogICAgZmkKICAgIGZvciBpbmRleCBpbiAiJHshcmVxdWlyZWRfdmFyc1tAXX0i OyBkbwogICAgICBrZXk9IiR7cmVxdWlyZWRfdmFyc1skaW5kZXhdfSIKICAgICAganFfcXV lcnk9Ii5yYWcuJHtrZXksLH0gfCBzZWxlY3QgKC4hPW51bGwpIgogICAgICB2YWx1ZT0kG VjaG8gIiR7Q09ORklHX0pTT059IiB8IGpxIC1yICIke2pxX3F1ZXJ5fSIpCiAgICAgIGlmI FtbIC16ICIke3ZhbHVlfSIgXV07IHRoZW4gCiAgICAgICAgZXJyb3JfZXhpdCAiJHtrZXl9 IGlzIHJlcXVpcmVkIGJ1dCBub3Qgc2V0LiIKICAgICAgZWxzZQogICAgICAgIGV2YWwgJHt rZXl9PVwiIiR7dmFsdWV9IlwiCiAgICAgICAgIGZpCiAgICBkb25lCgogICAgUkFHX1VSST0iJH tSQUdfUkVQT19OQU1FfS8ke1JBR19OQU1FfToke1JBR19WRVJTSU9OfSIKICAgIEVNQkVEX 01PREVMX1VSST0iJHtFTUJFRF9SRVBPX05BTUV9LyR7RU1CRURfTkFNRX06JHtFTUJFRF9W RVJTSU9OfSIKCiAgICBOR0NfQ0xJX1ZFUlNJT049IjMuNDEuMiIKICAgIE5HQ19DTElfVVJ MPSJodHRwczovL2FwaS5uZ2MubnZpZGlhLmNvbS92Mi9yZXNvdXJjZXMvbnZpZGlhL25nYy

Tableau 3-6. Image du conteneur NVIDIA RAG (suite)

Composant	Description

1hcHBzL25nY19jbGkvdmVyc2lvbnMvJHtOR0NfQ0xJX1ZFUlNJT059L2ZpbGVzL25nY2Nsa
V9saW51eC56aXAiCgogICAgbWtkaXIgLXAgL29wdC9kYXRhCiAgICBjZCAvb3B0L2RhdGEK
CiAgICBpZiBbICEgLWYgLmZpbGVfZG93bmxvYWRlZCBdOyB0aGVuCiAgICAgICMgY2xlYW4
gdXAKICAgICAgcm0gLXJmIGNvbXBvc2UuZW52ICR7UkFHX05BTUV9KiAke0xMTV9OQU1FfS
ogbmdjKiAke0VNQkVEX05BTUV9KiAqLmpzb24gLmZpbGVfZG93bmxvYWRlZAoKICAgICAgI
yBpbnN0YWxsIG5nYy1jbGkKICAgICAgd2dldCAtLWNvbnRlbnQtZGlzcG9zaXRpb24gJHtO
R0NfQ0xJX1VSTH0gLU8gbmdjY2xpX2xpbnV4LnppcCAmJiB1bnppcCBuZ2NjbGlfbGludXg
uemlwCiAgICAgIGV4cG9ydCBQQVRIPWBwd2RgL25nYy1jbGk6JHtQQVRIfQoKICAgICAgQV
BJS0VZPSIiCiAgICAgIFJFR19VUkk9Im52Y3IuaW8iCgogICAgICBpZiBbWyAiJChncmVwI
HJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2
YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpIiA9PSAqIiR7UkVHX1VSSX0iKiBdXTsgdGhlbgo
gICAgICAgIEFQSUtFWT0kKGdyZXAgcmVnaXN0cnktcGFzc3dkIC9vcHQvZGx2bS9vdmYtZW
52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKICAgI
CAgZmkKCiAgICAgIGlmIFsgLXogIiR7QVBJS0VZfSIgXTsgdGhlbgogICAgICAgICAgZXJy
b3JfZXhpdCAiTm8gQVBJS0VZIGZvdW5kIgogICAgICBmaQoKICAgICAgIyBjb25maWcgbmd
jLWNsaQogICAgICBta2RpciAtcCB+Ly5uZ2MKICAgICAgIGNhdCA8PCBFT0YgPiB+Ly5uZ2
MvY29uZmlnCiAgICAgIFtDVVJSRU5UXQogICAgICBhcGlrZXkgPSAke0FQSUtFWX0KICAgI
CAgZm9ybWF0X3R5cGUgPSBhc2NpaQogICAgICBvcmcgPSAke09SR19OQU1FfQogICAgICB0
ZWFtID0gJHtPUkdfVEVBTV9OQU1FfQogICAgICBhY2UgPSBuby1hY2UKICAgICAgIEVPRgoKICA
gICAgIyBuZ2MgZG9ja2VyIGxvZ2luCiAgICAgIGRvY2tlciBsb2dpbiBudmNyLmlvIC11IF
wkb2F1dGh0b2tlbiAtcCAke0FQSUtFWX0KCiAgICAgICMgZG9ja2VyaHViIGxvZ2luIGZvc
iBnZW5lcmFsIGNvbXBvbmVudHMsIGUuZy4gbWluaW8KICAgICAgRE9DS0VSSFVCX1VSST0k
KGdyZXAgcmVnaXN0cnktMi11cmkgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICCd
zLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQogICAgICBET0NLRVJIVUJfVVNFUk
5BTUU9JChncmVwIHJlZ2lzdHJ5LTItdXNlciAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZ
WQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCiAgICAgIERPQ0tFUkhV
Ql9QQVNTV09SRD0kKGdyZXAgcmVnaXN0cnktMi1wYXNzd2QgL29wdC9kbHZtL292Zi1lbnY
ueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIC
AgaWYgW1sgLW4gIiR7RE9DS0VSSFVCX1VTRVJOQU1FfSIgJiYgLW4gIiR7RE9DS0VSSFVCX
1BBU1NXT1JEfSIgXV07IHRoZW4KICAgICAgICBkb2NrZXIgbG9naW4gLXUgJHtET0NLRVJI
VUJfVVNFUk5BTUV9IC1wICR7RE9DS0VSSFVCX1BBU1NXT1JEfQogICAgICBlbHNlCiAgICA
gICAgZWNobyAiV2FybmluZzogRG9ja2VySHViIG5vdCBsb2dpbiIKICAgICAgZmkKCiAgIC
AgICMgZ2V0IFJBRyBmaWxlcwogICAgICBuZ2MgcmVnaXN0cnkgcmVzb3VyY2UgZG93bmxvY
WQtdmVyc2lvbiAke1JBR19VUkl9CgogICAgICAjIGdldCBsbG0gbW9kZWwKICAgICAgaWYg
WyAiJHtJTkZUUkVOQ0VfVFlQRX0iID0gInRydCIgXTsgdGhlbgogICAgICAgIExMTV9NT0R
FTF9VUkk9IiR7TExNX1JFUE9fTkFNRX0vJHtMTE1fTkFNRX06JHtMTE1fVkVSU0lPTn0iCi
AgICAgICAgbmdjIHJlZ2lzdHJ5IG1vZGVsIGRvd25sb2FkLXZlcnNpb24gJHtMTE1fTU9ER
UxfVVJJfQogICAgICAgIGNobW9kIC1SIG8rclggJHtMTE1fTkFNRX1fdiR7TExNX1ZFUlNJ
T059CiAgICAgICAgTExNX01PREVMX0ZPTERFUj0iL29wdC9kYXRhLyR7TExNX05BTUV9X3Y
ke0xMTV9WRVJTSU9OfSIKICAgICAgZWxpZiBbICIke0lORkVSRU5DRV9UWVBFfSIgPSAidm
xsbSIgXTsgdGhlbgogICAgICAgIHBpcCBpbnN0YWxsIGh1Z2dpbmdmYWNlX2h1YgogICAgI
CAgIGh1Z2dpbmdmYWNlLWNsaSBsb2dpbiAtLXRva2VuICR7SEZfVE9LRU59CiAgICAgICAg
aHVnZ2luZ2ZhY2UtY2xpIGRvd25sb2FkIC0tcmVzdW1lLWRvd25sb2FkICR7SEZfUkVQT30
vJHtMTE1fTkFNRX0gLS1sb2NhbC1kaXIgJHtMTE1fTkFNRX0gLS1sb2NhbC1kaXItdXNlLX
N5bWxpbmtzIEZhbHNlCiAgICAgICAgTExNX01PREVMX0ZPTERFUj0iL29wdC9kYXRhLyR7T
ExNX05BTUV9IgogICAgICAgIGNhdCA8PCBFT0YgPiAke0xMTV9NT0RFTF9GT0xERVJ9L21v
ZGVsX2NvbmZpZy55YW1sIAogICAgICAgIGVuZ2luZToKICAgICAgICAgIG1vZGVsOiAvbW9
kZWwtc3RvcmUKICAgICAgICAgIGVuZm9yY2VfZWFnZXI6IGZhbHNlCiAgICAgICAgICBtYX
hfY29udGV4dF9sZW5fdG9fY2FwdHVyZTogODE5MgogICAgICAgICAgbWF4X251bV9zZXFzO
iAyNTYKICAgICAgICAgIGR0eXBlOiBmbG9hdDE2CiAgICAgICAgICB0ZW5zb3JfcGFyYWxs
ZWxfc2l6ZTogJHtOVU1fR1BVfQogICAgICAgICAgZ3B1X21lbW9yeV91dGlsaXphdGlvbjo
gMC44CiAgICBFT0YKICAgICAgICBjaG1vZCAtUiBvK3JYICR7TExNX01PREVMX0ZPTERFUn
0KICAgICAgICBweXRob24zIC1jICJpbXBvcnQgeWFtbCwganNvbiwgc3lzOyBwcmludChqc
29uLmR1bXBzKHlhbWwuc2FmZV9sb2FkKHN5cy5zdGRpbi5yZWFkKCkpKSkiIDwgIiR7UkFH
X05BTUV9X3Yke1JBR19WRVJTSU9OfS9yYWctYXBwLXRleHQtY2hhdGJvdC55YW1sIj4gcmF
nLWFwcC10ZXh0LWNoYXRib3QuanNvbgogICAgICAgIGpxICcuc2VydmljZXMuIm5lbW9sbG
0taW5mZXJlbmNlIi5pbWFnZSA9ICJudmNyLmlvL252aWRpYS9uaW0vbmltX2xsbToyNC4wM
i1kYXkwIiB8CiAgICAgICAgICAgICAgIC5zZXJ2aWNlcy4ibmVtb2xsbS1pbmZlcmVuY2UiLmNv
bW1hbmQgPSAibmltX3ZsbG0gLS1tb2RlbF9uYW1lICR7TU9ERUxfTkFNRX0gLS1tb2RlbF9
jb25maWcgL21vZGVsLXN0b3JlL21vZGVsX2NvbmZpZy55YW1sIiB8CiAgICAgICAgICAgIC
5zZXJ2aWNlcy4ibmVtb2xsbS1pbmZlcmVuY2UiLnBvcnRzICs9IFsiODAwMDo4MDAwIl0gf

Tableau 3-6. Image du conteneur NVIDIA RAG (suite)

Composant	Description
	AogICAgICAgICAgICAuc2VydmljZXMuIm5lbW9sbG0taW5mZXJlbmNlIi5leHBvc2UgKz0g WyI4MDAwIl0nIHJhZy1hcHAtdGV4dC1jaGF0Ym90Lmpzb24gPiB0ZW1wLmpzb24gJiYgbXY gdGVtcC5qc29uIHJhZy1hcHAtdGV4dC1jaGF0Ym90Lmpzb24KICAgICAgICBweXRob24zIC 1jICJpbXBvcnQgeWFtbCwganNvbiwgc3lzOyBwcmludCh5YW1sLnNhZmVfZHVtcChqc29uL mxvYWQoc3lzLnN0ZGluKSwgZGVmYXVsdF9mbG93X3N0eWxlPUZhbHNlLCBzb3J0X2tleXM9 RmFsc2UpKSIgPCByYWctYXBwLXRleHQtY2hhdGJvdC5qc29uID4gIiR7UkFHX05BTUV9X3Y ke1JBR19WRVJTSU9fS9yYWctYXBwLXRleHQtY2hhdGJvdC55YW1sIgogICAgICBmaQoKIC AgICAgIyBnZXQgZW1iZWRkaW5nIG1vZGVscwogICAgICBuZ2MgcmVnaXN0cnkgbW9kZWwgZ G93bmxvYWQtdmVyc2lvbiAke0VNQkVEX01PREVMX1VSSX0KICAgICAgY2htb2QgLVIgbyty WCAke0VNQkVEX05BTUV9X3Yke0VNQkVEX1ZFUlNJT059CgogICAgICAjIGNvbmZpZyBjb21 wb3NlLmVudgogICAgICBjYXQgPDwgRU9GID4gY29tcG9zZS5lbnYKICAgICAgZXhwb3J0IE 1PREVMX0RJUkVDVE9SWT0iJHtMTE1fTU9ERUxfRk9MREVSfSIKICAgICAgZXhwb3J0IE1PR EVMX05BTUU9JHtMTE1fTkFNRX0KICAgICAgZXhwb3J0IE5VTV9HUFU9JHtOVU1fR1BVfQog ICAgICBleHBvcnQgQVBQX0NPTkZJR19GSUxFPS9kZXYvbnVsbAogICAgICBleHBvcnQgRU1 CRURESU5HX01PREVMX0RJUkVDVE9SWT0iL29wdC9kYXRhLyR7RU1CRURffTkFNRX1fdiR7RU 1CRURfVkVSU0lPTn0iCiAgICAgIGV4cG9ydCBFTUJFRERJTkdfTU9ERUxfTkFNRT0ke0VNQ kVEX1RZUEV9CiAgICAgIGV4cG9ydCBFTUJFRERJTkdfTU9ERUxfQ0tQVF9OQU1FPSIke0VN QkVEX1RZUEV9LSR7RU1CRURfVkVSU0lPTn0ubmVtbyIKICAgIEVPRgoKICAgICAgdG91Y2g gLmZpbGVfZG93bmxvYWRlZAogICAgZmkKCiAgICAjIHN0YXJ0IE5HQyBSQUcKICAgIGRvY2 tlciBjb21wb3NlIC1mICR7UkFHX05BTUV9X3Yke1JBR19WRVJTSU9OfS9kb2NrZXItY29tc G9zZS12ZWN0b3JkYi55YW1sIHVwIC1kIHBndmVjdG9yCiAgICBzb3VyY2UgY29tcG9zZS5l bnY7IGRvY2tlciBjb21wb3NlIC1mICR7UkFHX05BTUV9X3Yke1JBR19WRVJTSU9OfS9yYWc tYXBwLXRleHQtY2hhdGJvdC55YW1sIHVwIC1kCgotIHBhdGg6IC9vcHQvZGx2bS91dGlscy 5zaAogIHBlcm1pc3Npb25zOiAnMDc1NScKICBjb250ZW50OiB8CiAgICAjIS9iaW4vYmFza AogICAgZXJyb3JfZXhpdCgpIHsKICAgICAgZWNobyAiRXJyb3I6ICQxIiA+JjIKICAgICAg dm10b29sc2QgLS1jbWQgImluZm8tc2V0IGd1ZXN0aW5mby52bXNlcnZpY2UuYm9vdHN0cmF wLmNvbmRpdGlvbiBmYWxzZSwgREExXb3JrbG9hZEZhaWx1cmUsICQxIgogICAgICBleGl0ID EKICAgIH0KCiAgICBjaGVja19wcm90b2NvbCgpIHsKICAgICAgbG9jYWwgcHJveHlfdXJsP SQxCiAgICAgIHNoaWZ0CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIp CiAgICAgIGlmIFtbIC1uICIke3Byb3h5X3VybH0iIF1dOyB0aGVuCiAgICAgICAgbG9jYWw gcHJvdG9jb2w9JChlY2hvICIke3Byb3h5X3VybH0iIHwgYXdrIC1GICc6Ly8nICd7aWYgKE 5GID4gMSkgcHJpbnQgJDE7IGVsc2UgcHJpbnQgIiJ9JykKICAgICAgICBpZiBbIC16ICIkc HJvdG9jb2wiIF07IHRoZW4KICAgICAgICAgIGVjaG8gIk5vIHNwZWNpZmljIHByb3RvY29s IHByb3ZpZGVkLiBTa2lwcGluZyBwcm90b2NvbCBjaGVjay4iCiAgICAgICAgICAgICAgICB gMAogICAgICAgICAgICAgICAgICAgbG9jYWwgcHJvdG9jb2xfaW5jbHVkZWQ9ZmFsc2UKIC AgICAgICBmb3IgdmFyIGluICIke3N1cHBvcnRlZF9wcm90b2NvbHNbQF19IjsgZG8KICAgI CAgICAgICAgIFtbICIke3Byb3RvY29sfSIgPT0gIiR7dmFyfSIgXV07IHRoZW4KICAgICAg ICAgICAgICAgICAgICBicmVhawogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICA gICAgZmkKICAgICAgICBkb25lCiAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2xfaW5jbHVkZW R9IiA9PSBmYWxzZSBdXTsgdGhlbgogICAgICAgICAgICAgICAgZXJyb3JfZXhpdCAiVW5zdXBwb3J0Z WQgcHJvdG9jb2w6ICR7cHJvdG9jb2x9LiBTdXBwb3J0ZWQgcHJvdG9jb2xzIGFyZTogJHtz dXBwb3J0ZWRfcHJvdG9jb2xzWypdfSIKICAgICAgICBmaQogICAgICBmaQogICAgfQoKICA gICMgJEA6IGxpc3Qgb2Ygc3VwcG9ydGVkIHByb3RvY29scwogICAgc2V0X3Byb3h5KCkgew ogICAgICBsb2NhbCBzdXBwb3J0ZWRfcHJvdG9jb2xzPSgiJEAiKQoKICAgICAgQ09ORklHX 0pTT05fQkFTRTY0PSQoZ3JlcCAnY29uZmlnLWpzb24nIC9vcHQvZGx2bS9vdmYtZW52Lnht bCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKICAgICAgQ09 ORklHX0pTT049JChlY2hvICR7Q09ORklHX0pTT05fQkFTRTY0fSB8IGJhc2U2NCAtLWRlY2 9kZSkKICAgICAgIEhUVFBfUFJPWFlfVVJMPSQoZWNobyAiJHtDT05GSUdfSlNPTn0iIHwga nEgLXIgJy5odHRwX3Byb3h5IC8vIGVtcHR5JykKICAgICAgICAgSFRUUFNfUFJPWFlfVVJMPSQo ZWNobyAiJHtDT05GSUdfSlNPTn0iIHwganEgLXIgJy5odHRwc19wcm94eSAvLyBlbXB0eSc pCiAgICAgIGlmIFtbICQ/ IC1uZSAwIHx8ICgteiAiJHtIVFRQX1BST1hZX1VSTH0iICYmIC16ICIke0hUVFBTX1BST1h ZX1VSTH0iKSBdXTsgdGhlbgogICAgICAgIGluZm86IFRoZSBjb25maWctanNvbi B3YXMgcGFyc2VkLCBidXQgbm8gcHJveHkgc2V0dGluZ3Mgd2VyZSBmb3VuZC4iCiAgICAgI CAgcmV0dXJuIDAKICAgICAgZmkKCiAgICAgIGNoZWNrX3Byb3RvY29sICIke0hUVFBfUFJP WFlfVVJMfSIgIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iCiAgICAgIGNoZWNrX3Byb3R vY29sICIke0hUVFBTX1BST1hZX1VSTH0iICIke3N1cHBvcnRlZF9wcm90b2NvbHNbQF19Ig oKICAgICAgaWYgISBncmVwIC1xICdodHRwX3Byb3h5JyAvZXRjL2Vudmlyb25tZW50OyB0a GVuCiAgICAgICAgZWNobyAiZXhwb3J0IGh0dHBfcHJveHk9JHtIVFRQX1BST1hZX1VSTH0K ICAgICAgICBleHBvcnQgaHR0cHNfcHJveHk9JHtIVFRQU19QUk9YWV9VUkx9CiAgICAgICA gZXhwb3J0IEhUVFBfUFJPWFk9JHtIVFRQX1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSF

Tableau 3-6. Image du conteneur NVIDIA RAG (suite)

Composant	Description


```
RUUFNfUFJPWFk9JHtIVFRQU19QUk9YWV9VUkx9CiAgICAgICAgZXhwb3J0IG5vX3Byb3h5P
WxvY2FsaG9zdCwxMjcuMC4wLjEiID4+IC9ldGMvZW52aXJvbm1lbnQKICAgICAgICBzb3Vy
Y2UgL2V0Yy9lbnZpcm9ubWVudAogICAgICBmaQogICAgICAKICAgICAgIyBDb25maWd1cmU
gRG9ja2VyIHRvIHVzZSBhIHByb3h5CiAgICAgIG1rZGlyIC1wIC9ldGMvc3lzdGVtZC9zeX
N0ZW0vZG9ja2VyLnNlcnZpY2UuZAogICAgICBlY2hvICJbU2VydmljZV0KICAgICAgRW52a
XJvbm1lbnQ9XCJIVFRQX1BST1hZPSR7SFRUUF9QUk9YWV9VUkx9XCIKICAgICAgRW52aXJv
bm1lbnQ9XCJIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9
ubWVudD1cIk5PX1BST1hZPWxvY2FsaG9zdCwxMjcuMC4wLjFcIiIgPiAvZXRjL3N5c3RlbW
Qvc3lzdGVtL2RvY2tlci5zZXJ2aWNlLmQvcHJveHkuY29uZgogICAgICBzeXN0ZW1jdGwgZ
GFlbW9uLXJlbG9hZAogICAgICBzeXN0ZW1jdGwgcmVzdGFydCBkb2NrZXIKCiAgICAgIGVj
aG8gIkluZm86IGRvY2tlciBhbmQgc3lzdGVtIGVudmlyb25tZW50IGFyZSBub3cgY29uZml
ndXJlZCB0byB1c2UgdGhlIHByb3h5IHNldHRpbmdzIgogICAgfQ==
```


qui correspond au script suivant au format texte brut :

```
#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")
```

Tableau 3-6. Image du conteneur NVIDIA RAG (suite)

Composant	Description

```
    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}
```

Tableau 3-6. Image du conteneur NVIDIA RAG (suite)

Composant	Description

```
      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,
```

Tableau 3-6. Image du conteneur NVIDIA RAG (suite)

Composant	Description
	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" 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

Tableau 3-6. Image du conteneur NVIDIA RAG (suite)

Composant	Description
	<pre>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" }</pre> ■ Entrez les propriétés d'installation du pilote invité vGPU, telles que `vgpu-license` et `nvidia-portal-api-key`. ■ Fournissez les valeurs des propriétés requises pour un environnement déconnecté si nécessaire. Reportez-vous à la section Propriétés OVF des VM à apprentissage profond.
Sortie	■ Journaux d'installation du pilote invité vGPU dans `/var/log/vgpu-install.log`.

Tableau 3-6. Image du conteneur NVIDIA RAG (suite)

Composant	Description
	Pour vérifier que le pilote invité vGPU est installé, connectez-vous à la VM via SSH et exécutez la commande `nvidia-smi`. ■ Journaux de script cloud-init dans `/var/log/dl.log`. Pour suivre la progression du déploiement, exécutez `tail -f /var/log/dl.log`. ■ Exemple d'application Web d'agent conversationnel accessible à l'adresse `http://dl_vm_ip:3001/orgs/nvidia/models/text-qa-chatbot` Vous pouvez charger votre propre base de connaissances.

Attribuer une adresse IP statique à une VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA

Par défaut, les images de VM à apprentissage profond sont configurées avec l'attribution d'adresses DHCP. Pour déployer une VM à apprentissage profond avec une adresse IP statique directement sur un cluster vSphere, vous devez ajouter du code supplémentaire à la section cloud-init.

Sur vSphere with Tanzu, l'attribution d'adresses IP est déterminée par la configuration réseau du superviseur dans NSX.

Procédure

1 Créez un script cloud-init au format de texte brut pour la charge de travail DL que vous prévoyez d'utiliser.

Reportez-vous à la section Charges de travail d'apprentissage profond dans VMware Private AI Foundation with NVIDIA.

2 Ajoutez le code supplémentaire suivant au script cloud-init.

```
#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 Codez le script cloud-init obtenu au format base64.

4 Définissez le script cloud-init obtenu au format base64 comme valeur sur le paramètre OVF `user-data` de l'image de VM à apprentissage profond.

Exemple : Attribution d'une adresse IP statique à un exemple de charge de travail CUDA

Pour obtenir un exemple de VM à apprentissage profond avec une charge de travail DL Charges de travail d'apprentissage profond dans VMware Private AI Foundation with NVIDIA :

Élément VM à apprentissage profond	Valeur d'exemple
Image de charge de travail DL	nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8
Adresse IP	10.199.118.245
Préfixe de sous-réseau	/25
Passerelle	10.199.118.253
Serveurs DNS	■ 10.142.7.1 ■ 10.132.7.1

fournissez le code cloud-init suivant :

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBwLnNoCiAgcGVybWlzc2lvbnM6IC
cwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBkb2NrZXIgcnVuIC1kIG52Y3IuaW8vbnZpZGlhL2s4
cy9jdWRhLXNhbXBsZTp2ZWN0b3JhZGQtY3VkYTExLjcuMS11Ymk4CgptYW5hZ2VfZXRjX2hvc3RzOiB0cnVlCiAKd3JpdG
VfZmlsZXM6CiAgLSBwYXRoOiAvZXRjL25ldHdvcms55YW1sCiAgICBwZXJtaXNzaW9uczogJzA2
MDAnCiAgICBjb250ZW50OiB8CiAgICAgIG5ldHdvcms6CiAgICAgICAgdmVyc2lvbjogMgogICAgICAgIHJlbmRlcmVyOi
BuZXR3b3JrZAogICAgICAgIGV0aGVybmV0czoKICAgICAgICAgIGVuczMzOgogICAgICAgICAgICBkaGNwNDogZmFsc2Ug
IyBkaXNhYmxlIERIQ1A0CiAgICAgICAgICAgIGFkZHJlc3NlczogWzEwLjE5OS4xMTguMjQ1LzI1XSAgIyBTZXQgdGhlIH
N0YXRpYyBJUCBhZGRyZXNzIGFuZCBtYXNrCiAgICAgICAgICAgIHJvdXRlczoKICAgICAgICAgICAgICAgIC0gdG86IGRl
ZmF1bHQKICAgICAgICAgICAgICAgIHZpYTogMTE4LjI1MyAjIENvbmZpZ3VyZSBnYXRld2F5CiAgICAgICAgICAgICAgIC
AgICAgICAgICAgIG5hbWVzZXJ2ZXJzOgogICAgICAgICAgICAgICAgICAgIGFkZHJlc3NlczogWzEwLjE0Mi43LjEsIDEwLjEzMi43LjFdICMg
UHJvdmlkZSB0aGUgRE5TIHNlcnZlciBhZGRyZXNzLiBTZXBhcmF0ZSBtdWxpdHBsZSBETlMgc2VydmVyIGFkZHJlc3Nlcy
B3aXRoIGNvbW1hcy4KIApydW5jbWQ6CiAgLSBuZXRwbGFuIGFwcGx5
```

qui correspond au script suivant au format texte brut :

```
#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
```

Configurer une VM à apprentissage profond avec un serveur proxy

Pour connecter votre VM à apprentissage profond à Internet dans un environnement déconnecté où l'accès à Internet se fait sur un serveur proxy, vous devez fournir les détails du serveur proxy dans le fichier `config.json` de la machine virtuelle.

Procédure

1 Créez un fichier JSON avec les propriétés du serveur proxy.

Serveur proxy ne nécessitant aucune authentification	`{ "http_proxy": "protocol://ip-address-or-fqdn:port", "https_proxy": "protocol://ip-address-or-fqdn:port" }`
Serveur proxy nécessitant une authentification	`{ "http_proxy": "protocol://username:password@ip-address-or-fqdn:port", "https_proxy": "protocol://username:password@ip-address-or-fqdn:port" }`

Où :

- *protocol* est le protocole de communication utilisé par le serveur proxy, tel que `http` ou `https`.

- *username* et *password* sont les informations d'identification pour l'authentification sur le serveur proxy. Si ce dernier ne nécessite pas d'authentification, ignorez ces paramètres.
- *Ip-address-or-fqdn* : adresse IP ou nom d'hôte du serveur proxy.
- *Port* : numéro de port sur lequel le serveur proxy écoute les demandes entrantes.

2 Codez le code JSON résultant au format base64.

3 Lorsque vous déployez l'image de la VM à apprentissage profond, ajoutez la valeur codée à la propriété OVF `config-json`.

Dépannage du déploiement de VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA

Les informations de dépannage sur le déploiement d'une VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA fournissent des solutions aux problèmes potentiels que vous pourriez rencontrer.

- Automatisation de la charge de travail DL non effectuée

  Après le déploiement d'une VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA, la charge de travail DL spécifiée ne s'exécute pas.

- Le téléchargement d'une charge de travail DL échoue en raison d'informations d'identification d'authentification non valides

  Après le déploiement d'une VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA, le téléchargement de la charge de travail DL spécifiée sur la machine virtuelle échoue avec des messages de journal d'erreur indiquant des informations d'identification d'authentification non valides.

- Le téléchargement du pilote invité NVIDIA vGPU échoue en raison d'un lien de téléchargement manquant

  Après le déploiement d'une VM à apprentissage profond, le téléchargement du pilote invité vGPU spécifié sur la machine virtuelle échoue avec des messages de journal d'erreur indiquant qu'un lien ou une ressource de téléchargement est manquant.

- Le pilote invité NVIDIA vGPU s'affiche comme étant sans licence

  Une fois qu'une VM à apprentissage profond est déployée dans VMware Private AI Foundation with NVIDIA, l'état du pilote invité NVIDIA vGPU devient sans licence.

Automatisation de la charge de travail DL non effectuée

Après le déploiement d'une VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA, la charge de travail DL spécifiée ne s'exécute pas.

Problème

Déployez une VM à apprentissage profond avec une charge de travail DL à préinstaller lors du démarrage initial. Après le démarrage de la VM à apprentissage profond, la charge de travail DL ne s'effectue pas.

Cause

1 Le paramètre `user-data` codé en base64 ou les valeurs d'autres paramètres OVF, tels que `image-oneliner` ou `config-json` sont enregistrés ou décodés de manière incorrecte dans le fichier `/opt/dlvm/dl_app.sh`. Par conséquent, le script de charge de travail DL ne s'exécute pas.

2 L'installation du pilote vGPU a échoué, ce qui a entraîné la non-exécution du script cloud-init transmis dans le paramètre OVF `user-data`. Le script cloud-init repose sur l'installation réussie du pilote NVIDIA vGPU.

Solution

Sur la VM à apprentissage profond, vérifiez si la charge de travail DL est installée sur la machine virtuelle et appliquez une solution en conséquence.

Disponibilité de la charge de travail DL	Solution
Les composants de charge de travail DL ne sont pas créés sur la machine virtuelle.	■ Si vous utilisez un script cloud-init comme entrée dans le paramètre OVF `user-data`, vérifiez les valeurs suivantes : ■ Vérifiez le script codé et l'entrée en `user-data`. Assurez-vous que le paramètre `#cloud-config` s'affiche sur la première ligne et qu'il est inclus dans l'équivalent base64. ■ Vérifiez le `path`. ■ Vérifiez la chaîne codée en base64 et assurez-vous que la valeur `user-data` est correctement enregistrée dans `/opt/dlvm/dl_app.sh`. ■ Si vous utilisez d'autres paramètres OVF, vérifiez les valeurs suivantes : ■ `image-oneliner`. Vérifiez la chaîne codée en base64 et assurez-vous que la commande sur une ligne est correctement enregistrée dans `/opt/dlvm/dl_app.sh`. ■ `config-json`. Vérifiez la chaîne codée en base64 et assurez-vous que le fichier Docker Compose et `config.json`, s'ils sont fournis, sont correctement enregistrés dans `/root/docker-compose.yaml` et `/root/.docker/config.json`. Pour plus d'informations sur les paramètres OVF de la dernière image de VM à apprentissage profond, reportez-vous à la section Propriétés OVF des VM à apprentissage profond.
Les composants de charge de travail DL sont créés, mais la charge de travail n'est pas en cours d'exécution.	■ Vérifiez les messages d'erreur dans `/var/log/vgpu-install.log`. ■ Si vous utilisez un script cloud-init en entrée dans le paramètre OVF `user-data`, vérifiez si le pilote NVIDIA vGPU est installé et fonctionne correctement. Le script cloud-init n'est pas exécuté si l'installation du pilote NVIDIA vGPU échoue.

Le téléchargement d'une charge de travail DL échoue en raison d'informations d'identification d'authentification non valides

Après le déploiement d'une VM à apprentissage profond dans VMware Private AI Foundation with NVIDIA, le téléchargement de la charge de travail DL spécifiée sur la machine virtuelle échoue avec des messages de journal d'erreur indiquant des informations d'identification d'authentification non valides.

Problème

Si vous installez une image de conteneur de charge de travail DL, telle que Serveur d'inférence Triton, TensorFlow ou Pytorch, le fichier `/var/log/dl.log` contient le message suivant :

```
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>
```

Pour NVIDIA RAG, le fichier `/var/log/dl.log` contient le message suivant :

```
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
```

Cause

L'authentification sur le registre de conteneur nvcr.io a échoué. Par conséquent, vous ne pouvez pas télécharger l'image de charge de travail DL sur la machine virtuelle.

Solution

- Vérifiez les informations d'identification pour la connexion au registre nvcr.io transmises comme paramètres OVF ou à l'assistant de configuration du catalogue pour Private AI dans VMware Aria Automation.
  - Registre : nvcr.io
  - Compte d'utilisateur du registre : $oauthtoken
  - Mot de passe du registre : *NGC portal API key*
- Vérifiez que la clé API du portail NVIDIA NGC dispose des autorisations nécessaires pour accéder aux ressources requises et que la clé n'a pas expiré.

Le téléchargement du pilote invité NVIDIA vGPU échoue en raison d'un lien de téléchargement manquant

Après le déploiement d'une VM à apprentissage profond, le téléchargement du pilote invité vGPU spécifié sur la machine virtuelle échoue avec des messages de journal d'erreur indiquant qu'un lien ou une ressource de téléchargement est manquant.

Problème

Le fichier `/var/log/vgpu-install.log` contient l'un des messages suivants :

```
Erreur Aucun lien de téléchargement détecté via l'API
```

```
Aucun téléchargement trouvé via l'API
```

Cause

La clé API du portail de licences NVIDIA que vous transmettez en tant que valeur à la propriété OVF `nvidia-portal-api-key` ou à l'assistant de configuration du catalogue pour Private AI dans VMware Aria Automation n'est pas valide, a expiré ou n'est pas correctement formatée.

Solution

- Vérifiez que la clé API est valide.

- Vérifiez que la clé API est correctement entrée.

La clé API suit généralement le format UUID version 4 `xxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

Le pilote invité NVIDIA vGPU s'affiche comme étant sans licence

Une fois qu'une VM à apprentissage profond est déployée dans VMware Private AI Foundation with NVIDIA, l'état du pilote invité NVIDIA vGPU devient sans licence.

Problème

Le fichier `/var/log/vgpu-install.log` contient l'un des messages suivants :

```
État de la licence : sans licence
```

```
Sans licence (restreint)
```

Cause

Le jeton de configuration du client NVIDIA vGPU que vous transmettez comme valeur à la propriété OVF `vgpu-license` ou à l'assistant de configuration du catalogue pour Private AI dans VMware Aria Automation n'est pas valide, a expiré ou n'est pas correctement formaté.

Solution

- Vérifiez la validité du jeton de configuration du client.
- Vérifiez que la licence vGPU est correctement formatée et suit le format de jeton JWT, qui ressemble généralement à `eyxxxx.eyxxxxx.xxxxx`.

  Vous pouvez décoder le jeton JWT à jwt.io pour vérifier la date d'expiration et l'URL du serveur de nœud.
- Le jeton de licence vGPU est également enregistré dans `/etc/nvidia/ClientConfigToken/client_configuration_token.tok`.
- Pour résoudre le problème, exécutez cette commande pour rechercher des messages d'erreur spécifiques liés à la communication avec le dispositif NVIDIA License Server.

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

Pour appliquer un nouveau jeton, procédez comme suit :

1 Remplacez le contenu du fichier `/etc/nvidia/ClientConfigToken/client_configuration_token.tok` par un nouveau jeton, exécutez la commande suivante :

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

2 Redémarrez le service NVIDIA.

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

3 Vérifiez l'état de la licence du pilote invité NVIDIA vGPU.

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

Déploiement de charges de travail d'IA sur des clusters TKG dans VMware Private AI Foundation with NVIDIA

4

En tant qu'ingénieur DevOps, vous pouvez déployer des charges de travail d'IA de conteneur sur des clusters Tanzu Kubernetes Grid (TKG) dont les nœuds worker sont accélérés avec des GPU NVIDIA.

Pour plus d'informations sur la prise en charge de charges de travail d'IA sur les clusters TKG, reportez-vous à la section Déploiement de charges de travail d'IA/ML sur des clusters TKGS.

Lisez les sections suivantes :

- Provisionner un cluster TKG accéléré par GPU à l'aide d'un catalogue en libre-service dans VMware Private AI Foundation with NVIDIA
- Provisionner un cluster TKG accéléré par GPU à l'aide de la commande `kubectl` dans un environnement VMware Private AI Foundation with NVIDIA connecté
- Provisionner un cluster TKG accéléré par GPU à l'aide de la commande `kubectl` dans un environnement VMware Private AI Foundation with NVIDIA déconnecté

Provisionner un cluster TKG accéléré par GPU à l'aide d'un catalogue en libre-service dans VMware Private AI Foundation with NVIDIA

Dans VMware Private AI Foundation with NVIDIA, en tant qu'ingénieur DevOps, vous pouvez provisionner un cluster TKG accéléré avec des GPU NVIDIA à partir de VMware Aria Automation à l'aide d'éléments de catalogue en libre-service du cluster Kubernetes d'IA dans Automation Service Broker. Ensuite, vous pouvez déployer des images de conteneur d'IA à partir de NVIDIA NGC sur le cluster.

Conditions préalables

Vérifiez auprès de votre administrateur de cloud que VMware Private AI Foundation with NVIDIA est configuré. Reportez-vous à la section Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI.

Procédure

- Dans Automation Service Broker, déployez un élément de catalogue du cluster Kubernetes d'IA sur l'instance de superviseur configurée par l'administrateur de cloud.
  - Pour un cluster Kubernetes Tanzu Grid sans RAG, utilisez l'élément de catalogue **Cluster Kubernetes d'IA**. Reportez-vous à la section Déployer un cluster Tanzu Kubernetes Grid accéléré par GPU.
  - Pour un cluster de grille Tanzu Grid Kubernetes basé sur RAG, utilisez l'élément de catalogue **Cluster Kubernetes RAG d'IA**. Reportez-vous à la section Déployer un cluster RAG Tanzu Kubernetes Grid accéléré par GPU.

Étape suivante

Exécutez une image de conteneur d'IA. Dans un environnement connecté, utilisez le catalogue NVIDIA NGC. Dans un environnement déconnecté, utilisez le registre Harbor sur le superviseur.

Pour un cluster de grille Kubernetes Grid Tanzu basé sur RAG, déployez une base de données PostgreSQL pgvector dans VMware Data Services Manager et installez l'exemple de pipeline RAG à partir de NVIDIA. Reportez-vous à la section Déployer une charge de travail RAG sur un cluster TKG.

Provisionner un cluster TKG accéléré par GPU à l'aide de la commande `kubectl` dans un environnement VMware Private AI Foundation with NVIDIA connecté

Dans VMware Private AI Foundation with NVIDIA, en tant qu'ingénieur DevOps, provisionnez un cluster TKG qui utilise des GPU NVIDIA à l'aide de l'API Kubernetes. Ensuite, vous pouvez déployer des charges de travail d'IA de conteneur à partir du catalogue NVIDIA NGC.

Utilisez `kubectl` pour déployer le cluster TKG sur l'espace de noms configuré par l'administrateur de cloud.

Conditions préalables

Vérifiez auprès de l'administrateur de cloud que les conditions préalables suivantes sont en place pour l'infrastructure prête pour l'IA.

- VMware Private AI Foundation with NVIDIA est configuré. Reportez-vous à la section Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI.
- Dans un environnement déconnecté, une bibliothèque de contenu avec des images Ubuntu TKr est ajoutée à l'espace de noms vSphere pour les charges de travail d'IA. Reportez-vous à la section Configurer une bibliothèque de contenu avec Ubuntu TKr pour un environnement VMware Private AI Foundation with NVIDIA déconnecté.

Procédure

1 Connectez-vous au plan de contrôle du superviseur.

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

2 Provisionnez un cluster TKG et installez l'opérateur NVIDIA GPU et l'opérateur réseau NVIDIA sur celui-ci.

Reportez-vous à la section Workflow de l'opérateur de cluster pour le déploiement de charges de travail d'IA/ML sur des clusters TKGS.

Étape suivante

Déployez une image de conteneur d'IA à partir du catalogue NVIDIA NGC.

Provisionner un cluster TKG accéléré par GPU à l'aide de la commande `kubectl` dans un environnement VMware Private AI Foundation with NVIDIA déconnecté

Dans VMware Private AI Foundation with NVIDIA, en tant qu'ingénieur DevOps, provisionnez un cluster TKG qui utilise des GPU NVIDIA à l'aide de l'API Kubernetes. Dans un environnement déconnecté, vous devez également configurer un référentiel de modules Ubuntu local et utiliser le registre Harbor pour le superviseur.

Conditions préalables

Vérifiez auprès de l'administrateur de cloud que les conditions préalables suivantes sont en place pour l'infrastructure prête pour l'IA.

- VMware Private AI Foundation with NVIDIA est configuré pour un environnement déconnecté. Reportez-vous à la section Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI.
- Machine ayant accès au point de terminaison du superviseur et au référentiel Helm local hébergeant le dispositif pour les définitions de graphique de l'opérateur NVIDIA GPU.

Procédure

1 Provisionnez un cluster TKG sur l'espace de noms vSphere configuré par l'administrateur de cloud.

Reportez-vous à la section Provisionner un cluster TKGS pour NVIDIA vGPU.

2 Installez l'opérateur NVIDIA GPU.

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

3 Surveillez l'opération.

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

Étapes suivantes

Déployez une image de conteneur d'IA à partir du registre Harbor vers le superviseur.

Déploiement de charges de travail RAG dans VMware Private AI Foundation with NVIDIA

5

Une charge de travail de génération augmentée de récupération (RAG) se compose d'un grand modèle de langage (LLM, Large Language Model) et d'une base de connaissances externe avec les dernières données, stockés dans une base de données vectorielle. Dans VMware Private AI Foundation with NVIDIA, vous pouvez configurer une charge de travail RAG pour utiliser des intégrations à partir d'une base de données vectorielle gérée par VMware Data Services Manager.

Lisez les sections suivantes :

- Déployer une base de données vectorielle dans VMware Private AI Foundation with NVIDIA
- Déployer une VM à apprentissage profond avec une charge de travail RAG
- Déployer une charge de travail RAG sur un cluster TKG

Déployer une base de données vectorielle dans VMware Private AI Foundation with NVIDIA

Si vous prévoyez d'utiliser la génération augmentée de récupération (RAG) avec VMware Private AI Foundation with NVIDIA, configurez une base de données PostgreSQL avec pgvector à l'aide de VMware Data Services Manager.

Vous pouvez créer la base de données manuellement ou créer un catalogue en libre-service dans VMware Aria Automation qui peut être utilisé par des ingénieurs et des développeurs DevOps.

Conditions préalables

- Vérifiez que VMware Private AI Foundation with NVIDIA est disponible pour le domaine de charge de travail VI. Reportez-vous à la section Déploiement de VMware Private AI Foundation with NVIDIA.
- Vérifiez auprès de votre administrateur de cloud que les conditions préalables à la création d'une base de données PostgreSQL sont en place. Reportez-vous à la section Création de bases de données.
- Installez l'utilitaire de ligne de commande `psql` à partir du site Web PostgreSQL.

Procédure

1 Déployez une base de données PostgreSQL dans le domaine de charge de travail VI et obtenez la chaîne de connexion pour la base de données.

Vous pouvez utiliser l'un des workflows suivants. Si vous êtes scientifique des données, vous pouvez déployer directement une base de données à partir de VMware Aria Automation. Sinon, demandez un déploiement de bases de données à votre administrateur DSM ou à votre utilisateur DSM.

Workflow de déploiement.	Rôle d'utilisateur requis	Description
Déployer et obtenir la chaîne de connexion d'une base de données PostgreSQL à partir de VMware Aria Automation	Scientifique des données ou ingénieur DevOps	Reportez-vous à la section Déployer une base de données vectorielle à l'aide d'un élément de catalogue en libre-service dans VMware Aria Automation.
Déployez et obtenez la chaîne de connexion d'une base de données PostgreSQL à partir de la console VMware Data Services Manager.	Administrateur DSM ou utilisateur DSM, ou administrateur de cloud disposant de l'un de ces rôles	Reportez-vous aux sections Création de bases de données et Connexion à une base de données.
Déployer et obtenir la chaîne de connexion d'une base de données PostgreSQL à l'aide de la commande `kubectl`	Administrateur DSM ou utilisateur DSM, ou ingénieur DevOps disposant de l'un de ces rôles	Reportez-vous à la section Activation de la consommation en libre-service de VMware Data Services Manager.

Le format de la chaîne de connexion de la base de données déployée est le suivant.

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

2 Activez l'extension pgvector sur la base de données à l'aide de l'utilitaire de ligne de commande `psql`.

a Connectez-vous à la base de données.

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

b Activez l'extension pgvector.

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

Étape suivante

Intégrez la base de données dans votre charge de travail RAG. Reportez-vous aux sections Déployer une VM à apprentissage profond avec une charge de travail RAG et Déployer une charge de travail RAG sur un cluster TKG.

Déployer une base de données vectorielle à l'aide d'un élément de catalogue en libre-service dans VMware Aria Automation

Dans VMware Private AI Foundation with NVIDIA, en tant que scientifique des données ou ingénieur DevOps, vous pouvez déployer une base de données vectorielle à partir de VMware Aria Automation à l'aide d'un élément de catalogue en libre-service dans Automation Service Broker.

Procédure

1. Connectez-vous à VMware Aria Automation et, dans Automation Service Broker, recherchez l'élément du catalogue pour le déploiement de la base de données en fonction des informations de votre administrateur de cloud.

   Par défaut, l'élément de catalogue est appelé **DBaaS de DSM**.

2. Dans la carte de l'élément du catalogue, cliquez sur **Demande** et entrez les détails de la nouvelle base de données PostgreSQL.

   Pour plus d'informations sur les paramètres de la base de données, reportez-vous à la section Création de bases de données.

3. Obtenez la chaîne de connexion de la base de données déployée.
   a. Dans Automation Service Broker, cliquez sur **Déploiements > Déploiements**.
   b. Sélectionnez l'entrée de déploiement pour la base de données.
   c. Dans l'onglet **Topologie**, sélectionnez le modèle de cloud pour le déploiement de la base de données et, dans le menu **Actions** du modèle, sélectionnez **Obtenir la chaîne de connexion**.

Résultats

Pour plus d'informations sur le provisionnement et l'exécution d'opérations sur des bases de données dans VMware Data Services Manager à partir de VMware Aria Automation, reportez-vous au fichier `readme.md` dans le bundle `AriaAutomation_DataServicesManager` .

Déployer une VM à apprentissage profond avec une charge de travail RAG

Vous pouvez déployer une VM à apprentissage profond avec une charge de travail NVIDIA RAG à l'aide d'une base de données PostgreSQL pgvector gérée par VMware Data Services Manager.

Pour plus d'informations sur la charge de travail NVIDIA RAG, reportez-vous à la documentation Outil Docker Compose des applications NVIDIA RAG (nécessite des autorisations de compte spécifiques).

Conditions préalables

- Vérifiez que VMware Private AI Foundation with NVIDIA est configuré. Reportez-vous à la section Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI.
- Déployer une base de données vectorielle dans VMware Private AI Foundation with NVIDIA.

Procédure

1. Si, en tant que scientifique des données, vous déployez la VM à apprentissage profond à l'aide d'un élément de catalogue dans VMware Aria Automation, fournissez les détails de la base de données PostgreSQL pgvector après avoir déployé la machine virtuelle.
   a. Déployer un poste de travail RAG dans VMware Aria Automation.
   b. Accédez à **Consommer > Déploiements > Déploiements** et recherchez le déploiement de VM à apprentissage profond.
   c. Dans la section **VM Workstation**, enregistrez les détails de la connexion par SSH à la machine virtuelle.
   d. Connectez-vous à la VM à apprentissage profond via SSH à l'aide des informations d'identification disponibles dans Automation Service Broker.
   e. Ajoutez les variables pgvector suivantes au fichier `/opt/data/compose.env` :

```
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. Redémarrez l'application à conteneurs multiples NVIDIA RAG en exécutant les commandes suivantes.

      Par exemple, pour 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 Si, en tant qu'ingénieur DevOps, vous déployez la VM à apprentissage profond pour un scientifique des données directement sur le cluster vSphere ou à l'aide de la commande `kubectl`, créez un script cloud-init et déployez la VM à apprentissage profond.

   a Créez un script cloud-init pour NVIDIA RAG et la base de données PostgreSQL pgvector que vous avez créée.

   Vous pouvez modifier la version initiale du script cloud-init pour NVIDIA RAG. Par exemple, pour NVIDIA RAG 24.03 et une base de données PostgreSQL pgvector avec les détails de connexion `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 Codez le script cloud-init au format base64.

Utilisez un outil de codage au format base64, tel que https://decode64base.com/ pour générer la version codée de votre script cloud-init.

c Déployez la VM à apprentissage profond, en transmettant la valeur base64 du script cloud-init au paramètre d'entrée `user-data`.

Voir Déployer une VM à apprentissage profond directement sur un cluster vSphere de VMware Private AI Foundation with NVIDIA ou Déployer une VM à apprentissage profond à l'aide de la commande kubectl dans VMware Private AI Foundation with NVIDIA.

Déployer une charge de travail RAG sur un cluster TKG

En tant qu'ingénieur DevOps, vous pouvez déployer sur un cluster TKG dans un superviseur une charge de travail RAG basée sur l'exemple de pipeline RAG de NVIDIA qui utilise une base de données PostgreSQL pgvector gérée par VMware Data Services Manager.

Conditions préalables

- Vérifiez que VMware Private AI Foundation with NVIDIA est disponible pour le domaine de charge de travail VI. Reportez-vous à la section Chapitre 2 Préparation de VMware Cloud Foundation pour le déploiement de charges de travail Private AI.
- Déployer une base de données vectorielle dans VMware Private AI Foundation with NVIDIA.

Procédure

1 Provisionnez un cluster TKG accéléré par GPU.

Vous pouvez utiliser l'un des workflows suivants.

Workflow de provisionnement	Étapes
En utilisant un élément de catalogue dans VMware Aria Automation	Déployer un cluster RAG Tanzu Kubernetes Grid accéléré par GPU.
En utilisant la commande `kubectl`	1 Provisionnez un cluster TKG accéléré par GPU à l'aide de la commande `kubectl`. ■ Pour un environnement connecté, reportez-vous à la section Provisionner un cluster TKG accéléré par GPU à l'aide de la commande kubectl dans un environnement VMware Private AI Foundation with NVIDIA connecté. ■ Pour un environnement déconnecté, reportez-vous à la section Provisionner un cluster TKG accéléré par GPU à l'aide de la commande kubectl dans un environnement VMware Private AI Foundation with NVIDIA déconnecté. 2 Installez l'opérateur LLM RAG. Reportez-vous à la section Installer l'opérateur LLM RAG.

2 Si vous avez utilisé la commande `kubectl` pour provisionner le cluster TKG, installez NVIDIA RAG LLM Operator sur le cluster TKG.

   Reportez-vous à la section Installer l'opérateur LLM RAG.

   Lors du déploiement, l'élément de catalogue **Cluster Kubernetes RAG d'IA** dans VMware Aria Automation installe automatiquement NVIDIA RAG LLM Operator sur le cluster TKG.

3 Téléchargez les manifestes de l'exemple de pipeline NVIDIA RAG.

   Reportez-vous à la section Exemple de pipeline RAG.

4 Configurez l'exemple de pipeline RAG avec la base de données PostgreSQL pgvector.

   a Modifiez le fichier YAML de l'exemple de pipeline.

      Reportez-vous à l'étape 4 de la section Exemple de pipeline RAG.

   b Dans le fichier YAML, configurez l'exemple de pipeline avec la base de données PostgreSQL pgvector à l'aide de la chaîne de connexion de la base de données.

      Reportez-vous à la section Base de données vectorielle pour l'exemple de pipeline RAG.

5 Afin de fournir une adresse IP externe pour l'exemple d'application de conversation, définissez `frontend.service.type` sur `loadBalancer` dans le fichier YAML.

6 Démarrez l'exemple de pipeline RAG.

   Reportez-vous à la section Exemple de pipeline RAG.

7 Pour accéder à l'exemple d'application de conversation, exécutez la commande suivante pour obtenir l'adresse IP externe de l'application.

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

8 Dans un navigateur Web, ouvrez l'exemple d'application de conversation à l'adresse **http://*application_external_ip*:3001/orgs/nvidia/models/text-qa-chatbot**.

Surveillance de VMware Private AI Foundation with NVIDIA

6

Vous pouvez surveiller les mesures de GPU au niveau du cluster et de l'hôte dans vSphere Client et VMware Aria Operations.

Dans VMware Aria Operations, vous pouvez surveiller les mesures de GPU au niveau des propriétés du cluster, du système hôte et de l'hôte. Pour plus d'informations, reportez-vous à Tableaux de bord Private AI (GPU) et Propriétés des composants de vCenter Server dans VMware Aria Operations.

Dans vSphere Client, vous pouvez surveiller les mesures de GPU de la manière suivante :

- Au niveau de l'hôte. Reportez-vous à la section Diagrammes de performances des hôtes dans vSphere.
- Au niveau du cluster dans des graphiques personnalisés. Reportez-vous à la section Utilisation des diagrammes avancés et personnalisés dans vSphere.