Guida di VMware Private AI Foundation with NVIDIA

23 LUG 2024

VMware Cloud Foundation 5.2

vmware by Broadcom

È possibile trovare la documentazione tecnica più aggiornata sul sito Web di VMware by Broadcom all'indirizzo:

https://docs.vmware.com/it/

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

Sommario

Informazioni sulla Guida di VMware Private AI Foundation with NVIDIA

La *Guida di VMware Private AI Foundation with NVIDIA* fornisce una panoramica dei componenti di VMware Private AI Foundation with NVIDIA e workflow generali per i casi d'uso di sviluppo e produzione.

Destinatari

Le informazioni contenute nella *Guida di VMware Private AI Foundation with NVIDIA* sono destinate agli amministratori del cloud, ai data scientist e ai tecnici DevOps dei data center che hanno familiarità con:

- Amministratori del cloud
  - Concetti relativi alla virtualizzazione e ai data center definiti dal software (SDDC)
  - Componenti hardware come switch top-of-rack (ToR) e switch inter-rack, server con storage collegato diretto, cavi e alimentatori
  - Metodi per la configurazione di NVIDIA GPU nei server in un data center
  - Utilizzo di VMware vSphere® per l'uso delle macchine virtuali.
  - Utilizzo di vSphere IaaS control plane per configurare e assegnare risorse di vSphere agli spazi dei nomi di vSphere in un supervisore.

  In qualità di amministratore del cloud, vedere le informazioni seguenti:

  - Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI
  - Capitolo 3 Distribuzione di una macchina virtuale di deep learning in VMware Private AI Foundation with NVIDIA
  - Capitolo 6 Monitoraggio di VMware Private AI Foundation with NVIDIA
- Data scientist
  - Container, inclusi Docker, grafici Helm e registro Harbor

  In qualità di data scientist, vedere le informazioni seguenti:

  - Capitolo 3 Distribuzione di una macchina virtuale di deep learning in VMware Private AI Foundation with NVIDIA

    - Capitolo 5 Distribuzione di carichi di lavoro RAG in VMware Private AI Foundation with NVIDIA
- Tecnici DevOps
    - Provisioning delle macchine virtuali in vSphere tramite l'API Kubernetes.
    - Container, inclusi Docker, grafici Helm e registro Harbor
    - Utilizzo di vSphere IaaS control plane per il provisioning di macchine virtuali e cluster TKG (Tanzu Kubernetes Grid).

    In qualità di tecnico DevOps, vedere le informazioni seguenti:

    - Capitolo 4 Distribuzione dei carichi di lavoro AI nei cluster TKG in VMware Private AI Foundation with NVIDIA
    - Capitolo 5 Distribuzione di carichi di lavoro RAG in VMware Private AI Foundation with NVIDIA

Componenti software VMware

La funzionalità della soluzione VMware Private AI Foundation with NVIDIA è disponibile in diversi componenti software in base al proprio ruolo nell'organizzazione.

Ruolo utente di destinazione	Categoria software	Versioni del software supportate
Amministratori del cloud	Componenti distribuiti in VMware Cloud Foundation	Vedere Componenti di VMware in VMware Private AI Foundation with NVIDIA.
Data scientist	Componenti della macchina virtuale di deep learning	Vedere Note di rilascio di VMware Deep Learning VM.
Tecnici DevOps	Versioni di TK (TKr)	Vedere Note di rilascio delle versioni di VMware Tanzu Kubernetes.

Documentazione di VMware correlata

La soluzione VMware Private AI Foundation with NVIDIA include uno stack di prodotti e componenti software VMware. La documentazione per tali prodotti software è la seguente:

- Documentazione di VMware Cloud Foundation
- Documentazione di VMware vSphere e vSAN
- Documentazione di VMware vSphere IaaS Control Plane
- Documentazione di VMware Aria Automation
- Documentazione di VMware Aria Operations
- Documentazione di VMware Aria Suite Lifecycle
- Documentazione di VMware Data Services Manager

Glossario di VMware Cloud Foundation

Glossario di VMware Cloud Foundation definisce termini specifici per VMware Cloud Foundation.

Che cos'è VMware Private AI Foundation with NVIDIA?

1

Poiché è una soluzione che include più componenti, VMware Private AI Foundation with NVIDIA può essere utilizzato per eseguire carichi di lavoro di AI generativa utilizzando l'elaborazione accelerata di NVIDIA, nonché la gestione dell'infrastruttura virtuale e la gestione del cloud di VMware Cloud Foundation.

VMware Private AI Foundation with NVIDIA fornisce una piattaforma per il provisioning dei carichi di lavoro AI negli host ESXi con GPU NVIDIA. Inoltre, l'esecuzione dei carichi di lavoro AI basati su container NVIDIA GPU Cloud (NGC) è convalidata in modo specifico da VMware.

VMware Private AI Foundation with NVIDIA supporta due casi d'uso:

Caso d'uso di sviluppo

Gli amministratori del cloud e i tecnici DevOps possono eseguire il provisioning dei carichi di lavoro AI, tra cui RAG (Retrieval-Augmented Generation), sotto forma di macchine virtuali di deep learning. I data scientist possono utilizzare queste istanze di Deep Learning VM per lo sviluppo di AI. Vedere Informazioni sulle immagini delle macchine virtuali di deep learning in VMware Private AI Foundation with NVIDIA.

Caso d'uso di produzione

Gli amministratori del cloud possono fornire ai tecnici DevOps un ambiente di VMware Private AI Foundation with NVIDIA per il provisioning di carichi di lavoro AI pronti per la produzione nei cluster TKG (Tanzu Kubernetes Grid) in vSphere IaaS control plane.

Per informazioni sui componenti che fanno parte della soluzione VMware Private AI Foundation with NVIDIA e sulla loro architettura in VMware Cloud Foundation, vedere Architettura di sistema di VMware Private AI Foundation with NVIDIA.

Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI

2

In qualità di amministratore del cloud, è necessario distribuire software specifico e configurare i domini dei carichi di lavoro VI di destinazione in modo che i data scientist e i tecnici DevOps possano distribuire carichi di lavoro AI oltre a VMware Private AI Foundation with NVIDIA.

Componenti di VMware in VMware Private AI Foundation with NVIDIA

La funzionalità della soluzione VMware Private AI Foundation with NVIDIA è disponibile in diversi componenti software.

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

Per informazioni sull'architettura e i componenti di VMware Private AI Foundation with NVIDIA, vedere Architettura di sistema di VMware Private AI Foundation with NVIDIA.

Workflow di distribuzione per VMware Private AI Foundation with NVIDIA

La funzionalità di VMware Private AI Foundation with NVIDIA si basa su un set fondamentale di componenti con componenti aggiuntivi necessari per abilitare la distribuzione di uno dei seguenti tipi di carico di lavoro di IA:

- Deep Learning VM in generale
- Carichi di lavoro IA in un cluster TKG con accelerazione GPU in generale
- Carichi di lavoro RAG come Deep Learning VM o applicazioni di Deep Learning in cluster TKG con accelerazione GPU

  La distribuzione di un carico di lavoro RAG estende l'approccio generale per Deep Learning VM e i carichi di lavoro AI nei cluster TKG con la distribuzione di un database PostgreSQL pgvector e la configurazione dell'applicazione con il database pgvector.

In un ambiente disconnesso è necessario eseguire passaggi aggiuntivi per configurare e distribuire le appliance e fornire risorse in locale, in modo che i carichi di lavoro possano accedervi.

Ambiente connesso

Attività	Casi d'uso della distribuzione dei carichi di lavoro di AI	Passaggi
Rivedere l'architettura e i requisiti per la distribuzione di VMware Private AI Foundation with NVIDIA.	Tutte	■ Architettura di sistema di VMware Private AI Foundation with NVIDIA ■ Requisiti per la distribuzione di VMware Private AI Foundation with NVIDIA
Configurare un'istanza del servizio di licenza nel portale delle licenze NVIDIA e generare un token di configurazione client.		Guida per l'utente del sistema di licenze NVIDIA.
Generare una chiave API per l'accesso al catalogo NVIDIA NGC.		Pull ed esecuzione dei container aziendali NVIDIA AI
Creare una libreria di contenuti per le immagini di Deep Learning VM.	Distribuzione di una macchina virtuale di deep learning	Creazione di una libreria di contenuti con immagini di macchine virtuali di deep learning per VMware Private AI Foundation with NVIDIA
Abilitare vSphere IaaS control plane (in precedenza denominato vSphere with Tanzu).	Tutte	Configurazione di vSphere IaaS Control Plane per VMware Private AI Foundation with NVIDIA
Distribuzione Distribuire VMware Aria Automation utilizzando VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.	Tutte Necessaria se i data scientist e i tecnici DevOps distribuiranno i carichi di lavoro utilizzando elementi catalogo self-service in VMware Aria Automation.	1 Automazione del cloud privato per VMware Cloud Foundation 2 Configurazione di VMware Aria Automation per VMware Private AI Foundation with NVIDIA
Distribuire VMware Aria Operations utilizzando VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.	Tutte	Gestione di operazioni intelligenti per VMware Cloud Foundation.

Attività	Casi d'uso della distribuzione dei carichi di lavoro di AI	Passaggi
Distribuire VMware Data Services Manager	Distribuzione di un carico di lavoro RAG	1 Installazione e configurazione di VMware Data Services Manager Distribuire un'istanza di VMware Data Services Manager nel dominio di gestione. 2 Creazione di un elemento catalogo di un database vettore in VMware Aria Automation
Configurare una macchina che abbia accesso all'istanza del supervisore e che disponga di Docker, Helm e Kubernetes CLI Tools for vSphere.	Tutte Necessaria se i carichi di lavoro di AI verranno distribuiti direttamente utilizzando il comando `kubectl`.	Installazione di Kubernetes CLI Tools for vSphere

Ambiente disconnesso

Attività	Opzioni di distribuzione del carico di lavoro AI correlate	Passaggi
Rivedere i requisiti per la distribuzione di VMware Private AI Foundation with NVIDIA.	Tutte	■ Architettura di sistema di VMware Private AI Foundation with NVIDIA ■ Requisiti per la distribuzione di VMware Private AI Foundation with NVIDIA
Distribuire un'istanza di NVIDIA Delegated License Service.		Installazione e configurazione dell'appliance virtuale DLS È possibile distribuire l'appliance virtuale nello stesso dominio dei carichi di lavoro AI o nel dominio di gestione.
1 Registrare un'istanza di NVIDIA DLS nel portale delle licenze NVIDIA e associare e installare un server delle licenze in tale istanza. 2 Generare un token di configurazione client.		■ Configurazione di un'istanza del servizio ■ Gestione di licenze in un server delle licenze.
Creare una libreria di contenuti per le immagini di Deep Learning VM	Distribuzione di una macchina virtuale di deep learning	Creazione di una libreria di contenuti con immagini di macchine virtuali di deep learning per VMware Private AI Foundation with NVIDIA
Abilitare vSphere IaaS control plane (in precedenza denominato vSphere with Tanzu)	Tutte	Configurazione di vSphere IaaS Control Plane per VMware Private AI Foundation with NVIDIA

Attività	Opzioni di distribuzione del carico di lavoro AI correlate	Passaggi
■ Configurare una macchina che abbia accesso a Internet in cui sono installati Docker e Helm installati. ■ Configurare una macchina che abbia accesso a vCenter Server per il dominio del carico di lavoro VI, l'istanza del supervisore e il registro del container locale. La macchina deve disporre di Docker, Helm e Kubernetes CLI Tools for vSphere.		■ Distribuzione di un host Bastion ■ Installazione di Kubernetes CLI Tools for vSphere
Configurare una libreria di contenuti per le versioni di Tanzu Kubernetes (TKr) per Ubuntu	■ Distribuire un carico di lavoro RAG in un cluster TKG con accelerazione GPU ■ Distribuire carichi di lavoro di AI in un cluster TKG con accelerazione GPU	Configurazione di una libreria di contenuti con TKr Ubuntu per un ambiente di VMware Private AI Foundation with NVIDIA disconnesso
Configurare un servizio del registro Harbor nel supervisore.	Tutte Necessaria se i carichi di lavoro di AI verranno distribuiti in un supervisore in vSphere IaaS control plane In un ambiente senza vSphere IaaS control plane, per eseguire il pull delle immagini dei container in Deep Learning VM in esecuzione direttamente in un cluster vSphere, è necessario configurare un registro di un altro fornitore.	Configurazione di un registro Harbor privato in VMware Private AI Foundation with NVIDIA
Caricare i componenti degli operatori NVIDIA nell'ambiente.	■ Distribuire un carico di lavoro RAG in un cluster TKG con accelerazione GPU ■ Distribuire carichi di lavoro di AI in un cluster TKG con accelerazione GPU	Caricamento dei componenti di NVIDIA GPU Operator in un ambiente disconnesso

Attività	Opzioni di distribuzione del carico di lavoro AI correlate	Passaggi
Specificare una posizione da cui scaricare i driver guest della vGPU.	Distribuzione di una macchina virtuale di deep learning	Caricare in un server Web locale le versioni del driver guest della vGPU richieste scaricate dal portale delle licenze NVIDIA e un indice in uno dei formati seguenti: ■ File di indice `.txt` con un elenco dei file `.run` o `.zip` dei driver guest della 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` ■ Indice di directory nel formato generato dai server Web, ad esempio NGINX e i server Apache HTTP. I file dei driver della vGPU specifici della versione devono essere forniti come file `.zip`.
Caricare le immagini dei container NVIDIA NGC in un registro di container privato, ad esempio il servizio registro Harbor del supervisore.	Tutte In un ambiente senza vSphere IaaS control plane, per eseguire il pull delle immagini dei container in Deep Learning VM in esecuzione direttamente in un cluster vSphere, è necessario configurare un registro di un altro fornitore.	Caricamento delle immagini del container AI in un registro Harbor privato in VMware Private AI Foundation with NVIDIA
Distribuire VMware Aria Automation utilizzando VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.	Tutte Necessaria se i data scientist e i tecnici DevOps distribuiranno i carichi di lavoro utilizzando elementi catalogo self-service in VMware Aria Automation.	1 Automazione del cloud privato per VMware Cloud Foundation 2 Configurazione di VMware Aria Automation per VMware Private AI Foundation with NVIDIA

Attività	Opzioni di distribuzione del carico di lavoro AI correlate	Passaggi
Distribuire VMware Aria Operations utilizzando VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.	Tutte	Gestione di operazioni intelligenti per VMware Cloud Foundation
Distribuire VMware Data Services Manager	Distribuzione di un carico di lavoro RAG	1 Installazione e configurazione di VMware Data Services Manager Distribuire un'istanza di VMware Data Services Manager nel dominio di gestione. 2 Creazione di un elemento catalogo di un database vettore in VMware Aria Automation

Leggi i seguenti argomenti:

- Architettura di sistema di VMware Private AI Foundation with NVIDIA
- Requisiti per la distribuzione di VMware Private AI Foundation with NVIDIA
- Creazione di una libreria di contenuti con immagini di macchine virtuali di deep learning per VMware Private AI Foundation with NVIDIA
- Configurazione di vSphere IaaS Control Plane per VMware Private AI Foundation with NVIDIA
- Configurazione di una libreria di contenuti con TKr Ubuntu per un ambiente di VMware Private AI Foundation with NVIDIA disconnesso
- Configurazione di un registro Harbor privato in VMware Private AI Foundation with NVIDIA
- Caricamento dei componenti di NVIDIA GPU Operator in un ambiente disconnesso
- Configurazione di VMware Aria Automation per VMware Private AI Foundation with NVIDIA

Architettura di sistema di VMware Private AI Foundation with NVIDIA

VMware Private AI Foundation with NVIDIA viene eseguito in VMware Cloud Foundation aggiungendo supporto per i carichi di lavoro AI nei domini del carico di lavoro VI con provisioning di vSphere IaaS control plane eseguito tramite kubectl e VMware Aria Automation .

Figura 2-1. Architettura di esempio per VMware Private AI Foundation with NVIDIA

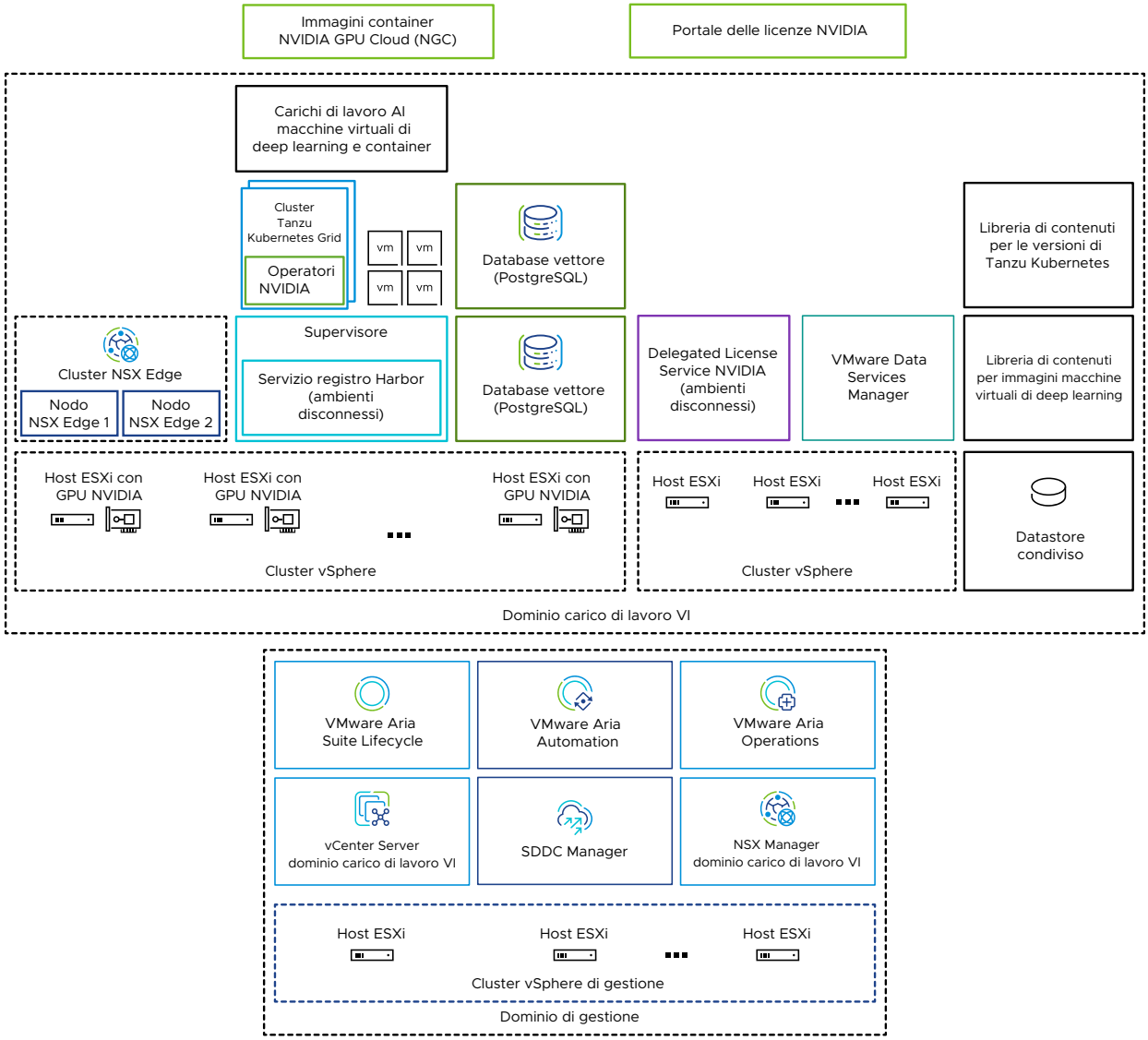


Tabella 2-1. Componenti per l'esecuzione dei carichi di lavoro AI in VMware Private AI Foundation with NVIDIA

Componente	Descrizione
Host ESXi abilitati per GPU	Host ESXi configurati nel modo seguente: ■ Dispongono di una GPU NVIDIA supportata per VMware Private AI Foundation with NVIDIA. La GPU viene condivisa tra i carichi di lavoro utilizzando il meccanismo di slicing temporale o MIG (Multi-Instance GPU). ■ Dispongono del driver NVIDIA vGPU Host Manager installato in modo che sia possibile utilizzare profili vGPU basati su MIG o slicing temporale.
Supervisore	Uno o più cluster vSphere abilitati per vSphere IaaS control plane in modo che sia possibile eseguire macchine virtuali e container in vSphere utilizzando l'API di Kubernetes. Un supervisore stesso è un cluster Kubernetes che funge da piano di controllo per gestire i cluster del carico di lavoro e le macchine virtuali.
Registro Harbor	Registro immagini locale in un ambiente disconnesso in cui vengono ospitate le immagini del container scaricate dal catalogo NVIDIA NGC.
Cluster NSX Edge	Cluster di nodi NSX Edge che fornisce il routing nord-sud a 2 livelli per il supervisore e i carichi di lavoro che esegue. Il gateway di livello 0 nel cluster NSX Edge è in modalità attivo-attivo.
Operatori NVIDIA	■ NVIDIA GPU Operator. Automatizza la gestione di tutti i componenti software NVIDIA necessari per eseguire il provisioning della GPU nei container in un cluster Kubernetes. NVIDIA GPU Operator viene distribuito in un cluster TKG. ■ NVIDIA Network Operator. Anche NVIDIA Network Operator consente di configurare i driver mellanox corretti per i container utilizzando funzioni virtuali per la rete ad alta velocità, RDMA e GPUDirect. L'operatore della rete collabora con l'operatore della GPU per abilitare RDMA di GPUDirect nei sistemi compatibili. NVIDIA Network Operator viene distribuito in un cluster TKG.
Database vettore	Database PostgreSQL in cui è abilitata l'estensione pgvector in modo che sia possibile utilizzarlo nei carichi di lavoro AI Retrieval Augmented Generation (RAG).

Tabella 2-1. Componenti per l'esecuzione dei carichi di lavoro AI in VMware Private AI Foundation with NVIDIA (continua)

Componente	Descrizione
■ Portale delle licenze NVIDIA ■ Delegated License Service (DLS) NVIDIA	Utilizzare il portale delle licenze NVIDIA per generare un token di configurazione client per assegnare una licenza al driver guest della vGPU nella macchina virtuale di deep learning e agli operatori della GPU nei cluster TKG. In un ambiente disconnesso o per fare in modo che i carichi di lavoro ricevano informazioni sulla licenza senza utilizzare una connessione Internet, ospitare localmente le licenze NVIDIA in un'appliance DLS (Delegated License Service).
Libreria di contenuti	Nelle librerie di contenuti vengono archiviate le immagini per le macchine virtuali di deep learning e per le versioni di Tanzu Kubernetes. Utilizzare queste immagini per la distribuzione dei carichi di lavoro AI nell'ambiente VMware Private AI Foundation with NVIDIA. In un ambiente connesso le librerie di contenuti estraggono i loro contenuti dalle librerie di contenuti pubbliche gestite da VMware. In un ambiente disconnesso è necessario caricare manualmente le immagini richieste o estrarle da un server mirror della libreria di contenuti interna.
Catalogo NVIDIA GPU Cloud (NGC)	Portale per container AI e ML ottimizzati per GPU che sono testati e pronti per l'esecuzione nelle GPU NVIDIA supportate in locale oltre a VMware Private AI Foundation with NVIDIA.

In qualità di amministratore del cloud, utilizzare i componenti di gestione in VMware Cloud Foundation

Tabella 2-2. Componenti di gestione in VMware Private AI Foundation with NVIDIA

Componente di gestione	Descrizione
SDDC Manager	Utilizzare SDDC Manager per le attività seguenti: ■ Distribuzione di un dominio del carico di lavoro VI abilitato per GPU basato sulle immagini di vSphere Lifecycle Manager e aggiunta di cluster in tale dominio. ■ Distribuzione di un cluster NSX Edge nei domini del carico di lavoro VI per l'utilizzo da parte delle istanze del supervisore e nel dominio di gestione per i componenti di VMware Aria Suite di VMware Private AI Foundation with NVIDIA. ■ Distribuzione di un'istanza di VMware Aria Suite Lifecycle integrata con il repository di SDDC Manager.
vCenter Server del dominio del carico di lavoro VI	Utilizzare questa istanza di vCenter Server per abilitare e configurare un supervisore.

Tabella 2-2. Componenti di gestione in VMware Private AI Foundation with NVIDIA (continua)

Componente di gestione	Descrizione
NSX Manager del dominio carico di lavoro VI	SDDC Manager utilizza questa istanza di NSX Manager per distribuire e aggiornare i cluster NSX Edge.
VMware Aria Suite Lifecycle	Utilizzare VMware Aria Suite Lifecycle per distribuire e aggiornare VMware Aria Automation e VMware Aria Operations.
VMware Aria Automation	Utilizzare VMware Aria Automation per aggiungere elementi catalogo self-service per la distribuzione dei carichi di lavoro AI per i tecnici DevOps e i data scientist.
VMware Aria Operations	Utilizzare VMware Aria Operations per monitorare l'utilizzo della GPU nei domini del carico di lavoro abilitati per GPU.
VMware Data Services Manager	Utilizzare VMware Data Services Manager per creare database vettore, ad esempio un database PostgreSQL con estensione pgvector.

Requisiti per la distribuzione di VMware Private AI Foundation with NVIDIA

Distribuire i componenti di VMware Private AI Foundation with NVIDIA nell'ambiente VMware Cloud Foundation in un dominio del carico di lavoro VI in cui devono essere installati determinati componenti NVIDIA.

Versioni del software VMware richieste

Vedere Componenti di VMware in VMware Private AI Foundation with NVIDIA.

Dispositivi GPU NVIDIA supportati

Prima di iniziare a utilizzare VMware Private AI Foundation with NVIDIA, assicurarsi che le GPU negli host ESXi siano supportate da VMware by Broadcom:

Tabella 2-3. Componenti NVIDIA supportati per VMware Private AI Foundation with NVIDIA

Componente NVIDIA	Opzioni supportate
GPU NVIDIA	■ NVIDIA A100 ■ NVIDIA L40S ■ NVIDIA H100
Modalità di condivisione GPU	■ Slicing temporale ■ Multi-Instance GPU (MIG)

Software NVIDIA richiesto

Il dispositivo GPU deve supportare i profili vGPU NVIDIA AI Enterprise (NVAIE) più recenti. Per istruzioni, vedere il documento GPU supportate da NVIDIA Virtual GPU Software.

- Driver host della vGPU NVIDIA (incluso il VIB per gli host ESXi) compatibile con la versione di VMware Cloud Foundation in uso. Vedere Note di rilascio di Virtual GPU Software per VMware vSphere.
- NVIDIA GPU Operator compatibile con la versione di Kubernetes dei cluster TKG distribuiti. Vedere Note di rilascio di NVIDIA GPU Operator e Note di rilascio delle versioni di VMware Tanzu Kubernetes.

Configurazione di VMware Cloud Foundation necessaria

Prima di distribuire VMware Private AI Foundation with NVIDIA, è necessario che in VMware Cloud Foundation sia disponibile una configurazione specifica.

- Licenza di VMware Cloud Foundation.
- Licenza del componente aggiuntivo VMware Private AI Foundation with NVIDIA.

  Per accedere alle funzionalità seguenti, è necessaria la licenza del componente aggiuntivo VMware Private AI Foundation with NVIDIA:

  - Configurazione di Private AI in VMware Aria Automation per gli elementi catalogo, per semplificare il provisioning delle macchine virtuali di deep learning con accelerazione GPU e dei cluster TKG.
  - Provisioning dei database PostgreSQL con l'estensione pgvector con il supporto Enterprise.
  - Distribuzione e utilizzo dell'immagine della macchina virtuale di deep learning fornita da VMware by Broadcom.

  È possibile distribuire carichi di lavoro AI con e senza un supervisore abilitato e utilizzare le metriche di GPU in vCenter Server e VMware Aria Operations con la licenza di VMware Cloud Foundation.
- Prodotto NVIDIA vGPU con licenza che include il file VIB del driver host per gli host ESXi e i driver del sistema operativo guest. Per istruzioni, vedere il documento GPU supportate da NVIDIA Virtual GPU Software.
- File VIB del driver host di NVIDIA vGPU scaricato da https://nvid.nvidia.com/
- Immagine di vSphere Lifecycle Manager con il file VIB del driver vGPU Host Manager disponibile in SDDC Manager. Vedere Gestione delle immagini di vSphere Lifecycle Manager in VMware Cloud Foundation.

- Dominio del carico di lavoro VI con almeno 3 host ESXi abilitati per GPU, basato sull'immagine di vSphere Lifecycle Manager contenente il file VIB del driver Host Manager. Vedere Distribuzione di un dominio del carico di lavoro VI tramite l'interfaccia utente di SDDC Manager e Gestione delle immagini di vSphere Lifecycle Manager in VMware Cloud Foundation.
- Driver host NVIDIA vGPU installato e vGPU configurata in ogni host ESXi del cluster per i carichi di lavoro AI.
  a In ogni host ESXi, abilitare SR-IOV nel BIOS e Shared Direct nei dispositivi grafici per le operazioni di AI.

    Per informazioni sulla configurazione di SR-IOV, vedere la documentazione del fornitore dell'hardware. Per informazioni sulla configurazione di Shared Direct nei dispositivi grafici, vedere Configurazione della grafica virtuale in vSphere.

  b Installare il driver NVIDIA vGPU Host Manager in ogni host ESXi in uno dei modi seguenti:
    - Installare il driver in ogni host e aggiungere il file VIB del driver all'immagine di vSphere Lifecycle per il cluster.

      Vedere Guida rapida del software NVIDIA Virtual GPU.
    - Aggiungere il file VIB del driver all'immagine di vSphere Lifecycle per il cluster e correggere gli host.

  c Se si desidera utilizzare la condivisione MIG (Multi-Instance GPU), abilitarla in ogni host ESXi nel cluster.

    Vedere Guida per l'utente di NVIDIA MIG.

  d Nell'istanza di vCenter Server per il dominio del carico di lavoro VI, impostare l'impostazione avanzata `vgpu.hotmigrate.enabled` su `true` in modo che le macchine virtuali con vGPU possano essere migrate tramite vSphere vMotion.

    Vedere Configurazione delle impostazioni avanzate.

Creazione di una libreria di contenuti con immagini di macchine virtuali di deep learning per VMware Private AI Foundation with NVIDIA

Le immagini di macchine virtuali di deep learning in VMware Private AI Foundation with NVIDIA vengono distribuite in una libreria di contenuti condivisa pubblicata da VMware. In qualità di amministratore del cloud, utilizzare una libreria di contenuti per estrarre immagini di macchine virtuali specifiche nel dominio del carico di lavoro VI durante la distribuzione delle macchine virtuali.

Prerequisiti

L'amministratore del cloud deve verificare che VMware Private AI Foundation with NVIDIA sia distribuito e configurato. Vedere Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI.

Procedura

1 Accedere all'istanza di vCenter Server per il dominio del carico di lavoro VI all'indirizzo **`https://<vcenter_server_fqdn>/ui`**.

2 Selezionare **Menu > Librerie di contenuti** e fare clic su **Crea**.

3 Creare una libreria di contenuti per le immagini delle macchine virtuali di deep learning.

- Per un ambiente connesso, creare una libreria di contenuti con sottoscrizione connessa a `https://packages.vmware.com/dl-vm/lib.json`. L'autenticazione non è necessaria.
- Per un ambiente disconnesso, scaricare le immagini delle macchine virtuali di deep learning da `https://packages.vmware.com/dl-vm/` e importarle in una libreria di contenuti locale.

  Per ogni immagine, scaricare i file `.ovf`, `.vmdk`, `.mf` e `.cert` pertinenti.

Configurazione di vSphere IaaS Control Plane per VMware Private AI Foundation with NVIDIA

Per consentire ai tecnici DevOps e ai data scientist di distribuire macchine virtuali di deep learning o cluster TKG con carichi di lavoro del container AI, è necessario distribuire un supervisore in un cluster abilitato per GPU in un dominio del carico di lavoro VI e creare classi di macchine virtuali abilitate per vGPU.

Prerequisiti

Vedere Requisiti per la distribuzione di VMware Private AI Foundation with NVIDIA.

Procedura

1 Distribuire un cluster NSX Edge nel dominio del carico di lavoro VI utilizzando SDDC Manager.

SDDC Manager distribuisce anche un gateway di livello 0 che viene specificato al momento della distribuzione del supervisore. Il gateway di livello 0 è in modalità ad alta disponibilità attiva-attiva.

2 Configurare un criterio di storage per il supervisore.

Vedere Creazione di criteri di storage per vSphere with Tanzu.

3 Distribuire un supervisore in un cluster di host ESXi abilitati per GPU nel dominio del carico di lavoro VI.

Utilizzare l'assegnazione dell'indirizzo IP statico per la rete di gestione. Assegnare la rete di gestione della macchina virtuale supervisore in vSphere Distributed Switch per il cluster.

Configurare la rete del carico di lavoro nel modo seguente:

- Utilizzare vSphere Distributed Switch per il cluster o crearne uno specifico per i carichi di lavoro di IA.
- Configurare il supervisore con il cluster NSX Edge e il gateway di livello 0 distribuiti utilizzando SDDC Manager.
- Impostare gli altri valori in base alla progettazione.

Utilizzare il criterio di storage creato.

Per ulteriori informazioni sulla distribuzione di un supervisore in un singolo cluster, vedere Distribuzione di un supervisore a una zona con rete NSX.

4 Configurare le classi di macchine virtuali basate su vGPU per i carichi di lavoro di IA.

In queste classi di macchine virtuali, impostare i requisiti di elaborazione e un profilo vGPU per un dispositivo NVIDIA GRID vGPU in base ai dispositivi vGPU configurati negli host ESXi nel cluster supervisore.

- Per informazioni sulla configurazione delle classi di macchine virtuali basate su vGPU per le macchine virtuali, vedere Creazione di una classe di macchine virtuali personalizzata tramite vSphere Client e Aggiunta di dispositivi PCI a una classe di macchine virtuali in vSphere with Tanzu.
- Per informazioni sulla configurazione di classi di macchine virtuali basate su vGPU per i nodi worker TKG, vedere Creazione di una classe di macchine virtuali personalizzata con un profilo vGPU in vSphere 8 Update 2b e versioni successive e Configurazione di spazi dei nomi vSphere per i cluster TKG nel supervisore.

Per la classe di macchine virtuali per la distribuzione di macchine virtuali di deep learning con carichi di lavoro NVIDIA RAG, impostare le seguenti impostazioni aggiuntive nella finestra di dialogo della classe di macchine virtuali:

- Selezionare il profilo vGPU completo per la modalità di slicing temporale o un profilo MIG. Ad esempio, per la scheda NVIDIA A100 a 40 GB in modalità di slicing temporale vGPU, selezionare **nvidia_a100-40c**.
- Nella scheda **Hardware virtuale** allocare più di 16 core CPU virtuali e 64 GB di memoria virtuale.
- Nella scheda **Parametri avanzati** impostare il parametro `pciPassthru<vgpu-id>.cfg.enable_uvm` su **1**.

dove `<vgpu-id>` identifica la vGPU assegnata alla macchina virtuale. Ad esempio, se alla macchina virtuale sono assegnate due vGPU, impostare `pciPassthru0.cfg.parameter=1` e `pciPassthru1.cfg.parameter = 1`.

5 Se si intende utilizzare lo strumento della riga di comando `kubectl` per distribuire una macchina virtuale di deep learning o un cluster TKG con accelerazione GPU in un supervisore, creare e configurare uno spazio dei nomi vSphere, aggiungendo limiti delle risorse, criterio di storage, autorizzazioni per i tecnici DevOps e associando le classi di macchine virtuali basate su vGPU a tale spazio dei nomi.

- Per informazioni sulla configurazione degli spazi dei nomi vSphere per le macchine virtuali, vedere Creazione e configurazione di uno spazio dei nomi vSphere nel supervisore.
- Per informazioni sulla configurazione degli spazi dei nomi vSphere per i cluster TKG, vedere Configurazione di spazi dei nomi vSphere per i cluster TKG nel supervisore.

6 Se si intende abilitare le distribuzioni di macchine virtuali di deep learning in un supervisore chiamando direttamente `kubectl`, aggiungere la libreria di contenuti allo spazio dei nomi di vSphere per i carichi di lavoro AI.

VMware Aria Automation crea uno spazio dei nomi ogni volta che viene eseguito il provisioning di una macchina virtuale di deep learning, aggiungendo automaticamente la libreria di contenuti in tale spazio dei nomi.

a Selezionare **Menu > Gestione carico di lavoro**.

b Passare allo spazio dei nomi per i carichi di lavoro AI.

c Nella scheda **Servizio macchina virtuale** fare clic su **Gestisci librerie di contenuti**.

d Selezionare la libreria di contenuti con le immagini delle macchine virtuali di deep learning e fare clic su **OK**.

Configurazione di una libreria di contenuti con TKr Ubuntu per un ambiente di VMware Private AI Foundation with NVIDIA disconnesso

Se l'ambiente non dispone di connettività Internet, l'amministratore del cloud fornisce una libreria di contenuti locale in cui può caricare manualmente le versioni di Tanzu Kubernetes (TKr) e associarle al supervisore.

La distribuzione di carichi di lavoro AI basati su NVIDIA nei cluster TKG richiede l'utilizzo della versione Ubuntu delle versioni di Tanzu Kubernetes.

Attenzione La libreria di contenuti TKr viene utilizzata in tutti gli spazi dei nomi vSphere nel supervisore quando si esegue il provisioning di nuovi cluster TKG.

Prerequisiti

L'amministratore del cloud deve verificare che VMware Private AI Foundation with NVIDIA sia distribuito e configurato. Vedere Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI.

Procedura

1 Scaricare le immagini TKr basate su Ubuntu con le versioni di Kubernetes richieste da `https://wp-content.vmware.com/v2/latest/`.

2 Accedere all'istanza di vCenter Server per il dominio del carico di lavoro VI all'indirizzo **`http://<vcenter_server_fqdn>/ui`**.

3 Creare una libreria di contenuti locale e importare le immagini TKr in tale libreria.

Vedere Creazione di una libreria di contenuti locale (per il provisioning del cluster air gap).

4 Aggiungere la libreria di contenuti al supervisore.

a Selezionare **Menu > Gestione carico di lavoro**.

b Passare al supervisore per i carichi di lavoro AI.

c Nella scheda **Configura** selezionare **Generale**.

d Accanto alla proprietà **Tanzu Kubernetes Grid Service** fare clic su **Modifica**.

e Nella pagina **Generale** visualizzata, espandere **Tanzu Kubernetes Grid Service** e accanto a **Libreria di contenuti** fare clic su **Modifica**.

f Selezionare la libreria di contenuti con le immagini TKr e fare clic su **OK**.

Configurazione di un registro Harbor privato in VMware Private AI Foundation with NVIDIA

È possibile utilizzare Harbor come servizio supervisore come registro locale per le immagini di container del catalogo NVIDIA NGC.

Nota L'installazione del servizio Harbor nel supervisore richiede una connessione Internet.

Se si desidera utilizzare l'integrazione del registro Harbor con il supervisore, è possibile seguire questi approcci di configurazione:

- Utilizzare un registro Harbor solo nel supervisore nel dominio del carico di lavoro abilitato per GPU. Eseguire le attività seguenti:

a Abilitazione di Harbor come servizio supervisore.

b Caricamento delle immagini del container AI in un registro Harbor privato in VMware Private AI Foundation with NVIDIA

È possibile disconnettere l'ambiente da Internet e iniziare a utilizzare il servizio Harbor come registro di container locale dopo aver installato il servizio o dopo averlo installato e aver scaricato il set iniziale di immagini di container richieste.

In questo approccio, è necessario scaricare manualmente le immagini di container dal catalogo NVIDIA NGC in una macchina nell'ambiente e quindi caricarle nel registro.

- Creazione di un registro Harbor in VMware Private AI Foundation with NVIDIA come replica di un registro connesso.

  Un registro Harbor in esecuzione al di fuori dell'ambiente VMware Private AI Foundation with NVIDIA è sempre connesso a Internet. Il registro Harbor nel supervisore per il dominio del carico di lavoro abilitato per GPU riceve le immagini di container dal registro connesso tramite un meccanismo proxy. In questo modo, i componenti principali dell'istanza di VMware Cloud Foundation rimangono isolati.

  In questo approccio sono necessarie risorse aggiuntive per il registro connesso.

Nota Allocare spazio di storage sufficiente per ospitare i container NVIDIA NGC che si intende distribuire in una macchina virtuale di deep learning o in un cluster TKG. Includere almeno tre versioni di ciascun container nello spazio di storage.

Se la propria organizzazione non consente la connessione a Internet durante l'installazione del servizio Harbor o la configurazione di un registro Harbor connesso, utilizzare un registro di container di un altro fornitore.

Caricamento delle immagini del container AI in un registro Harbor privato in VMware Private AI Foundation with NVIDIA

In un ambiente disconnesso in cui si utilizza un registro Harbor solo nel supervisore pronto per AI, è necessario caricare manualmente le immagini del container AI che si intende distribuire in una macchina virtuale di deep learning o in un cluster TKG dal catalogo NVIDIA NGC in Harbor.

Procedura

1. Nelle macchine per l'accesso a NVIDIA NGC e all'istanza di VMware Cloud Foundation disconnessa, configurare il client Docker con il certificato del registro Harbor.

   Vedere Configurazione del client Docker con un certificato del registro.

2. Accedere a NVIDA NGC.

   Utilizzare il nome utente riservato di $oauthtoken e incollare la chiave API nel campo della password.

   ```
   docker login nvcr.io
   ```

3 Estrarre le immagini del container richieste nella macchina con accesso al catalogo NVIDIA NGC e salvarle in un archivio.

   Ad esempio, per scaricare l'immagine del container di esempio CUDA, eseguire i comandi seguenti.

```
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 Copiare l'archivio nella macchina con accesso al registro del container locale.

5 Nella macchina con accesso al registro del container locale, caricare l'immagine del container.

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

6 Accedere al registro Harbor.

   Ad esempio, se il registro Harbor è in esecuzione in my-harbor-registry.example.com, eseguire i comandi seguenti.

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

7 Contrassegnare l'immagine di cui si desidera eseguire il push nel progetto con lo stesso nome dello spazio dei nomi in cui si desidera utilizzarla.

   Ad esempio, per contrassegnare l'immagine del container di esempio CUDA come la più recente per il progetto my-private-ai-namespace nel registro my-harbor-registry.example.com, eseguire il comando seguente.

```
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 Eseguire il push delle immagini del container nel registro Harbor.

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

Creazione di un registro Harbor in VMware Private AI Foundation with NVIDIA come replica di un registro connesso

Per poter eseguire facilmente l'aggiornamento alle immagini più recenti nel catalogo NVIDIA NGC, è possibile utilizzare un registro Harbor in un supervisore che si trova in un altro dominio del carico di lavoro VI o in un'altra istanza di VMware Cloud Foundation e può essere connesso a Internet. È quindi possibile replicare questo registro connesso nel supervisore in cui si intende eseguire i carichi di lavoro AI.

Estrarre le immagini del container più recenti da NVIDIA NGC nel registro Harbor connesso e trasferirle nel registro disconnesso utilizzando una connessione con proxy memorizzato nella cache. In questo modo, non è necessario scaricare di frequente le immagini del container e quindi caricarle manualmente.

Nota È inoltre possibile utilizzare un registro di container connesso di un altro fornitore.

Configurare la rete tra i due registri nel modo seguente:

- Il registro connesso può essere instradato al registro di replica.
- Il registro connesso viene inserito in una DMZ dove tra i due registri sono consentite solo le comunicazioni `docker push` e `docker pull`.

Prerequisiti

Abilitazione di Harbor come servizio supervisore nel supervisore nel dominio del carico di lavoro abilitato per GPU.

Procedura

1. Accedere all'interfaccia utente del registro Harbor connesso come amministratore di sistema di Harbor.
2. Passare alla pagina **Amministrazione > Registri** e creare un endpoint per il catalogo NVIDIA NGC `nvcr.io/nvaie` selezionando il provider **Registro Docker** e con la chiave NVIDIA NGC API.
3. Passare alla pagina **Amministrazione > Progetti** e creare un progetto con proxy memorizzato nella cache, connesso all'endpoint per `nvcr.io/nvaie`.
4. Nella pagina **Registri** creare un endpoint di replica per il registro disconnesso, selezionando il provider **Harbor**.
5. Passare alla pagina **Amministrazione > Repliche** e creare una regola di replica.
   - Utilizzare la modalità di replica basata su push.
   - Nella proprietà **Registro di destinazione** immettere l'URL del registro disconnesso nel supervisore pronto per AI.
   - Impostare filtri, spazio dei nomi di destinazione e modalità di attivazione in base ai requisiti dell'organizzazione.

Operazioni successive

1. Estrarre da NVIDIA NGC le immagini del container richieste dall'organizzazione nel registro connesso eseguendo `docker pull` nella macchina client Docker.
2. Se la regola di replica ha la modalità di attivazione manuale, eseguire le repliche manuali eventualmente necessarie.

Caricamento dei componenti di NVIDIA GPU Operator in un ambiente disconnesso

In un ambiente disconnesso, caricare i componenti di NVIDIA GPU Operator in posizioni interne.

Procedura

1. Fornire un repository di pacchetti Ubuntu locale e caricare le immagini del container nel pacchetto di NVIDIA GPU Operator nel registro Harbor per il supervisore.
2. Fornire un repository di grafici Helm locale con definizioni dei grafici di NVIDIA GPU Operator.
3. Aggiornare le definizioni del grafico Helm di NVIDIA GPU Operator per utilizzare il repository dei pacchetti Ubuntu locale e il registro Harbor privato.

Risultati

Per ulteriori informazioni, vedere Installazione di VMware vSphere with VMware Tanzu (air gap).

Configurazione di VMware Aria Automation per VMware Private AI Foundation with NVIDIA

VMware Aria Automation fornisce supporto per gli elementi catalogo self-service che i tecnici devOps e i data scientist possono utilizzare per eseguire il provisioning dei carichi di lavoro AI in VMware Private AI Foundation with NVIDIA in modo semplice e personalizzabile.

Prerequisiti

In qualità di amministratore del cloud, verificare che l'ambiente VMware Private AI Foundation with NVIDIA sia configurato. Vedere Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI.

Procedura

1. Connessione di VMware Aria Automation a un dominio del carico di lavoro per VMware Private AI Foundation with NVIDIA

   Prima di poter aggiungere elementi del catalogo per il provisioning delle applicazioni di intelligenza artificiale utilizzando VMware Aria Automation, connettere VMware Aria Automation a VMware Cloud Foundation.
2. Creazione di elementi catalogo self-service AI in VMware Aria Automation

   In qualità di amministratore del cloud, utilizzare la procedura guidata di configurazione del catalogo per Private AI in VMware Aria Automation per aggiungere rapidamente elementi catalogo per la distribuzione di istanze di Deep Learning VM o cluster TKG con accelerazione GPU in un dominio del carico di lavoro VI in VMware Cloud Foundation connesso.

3 Creazione di un elemento catalogo di un database vettore in VMware Aria Automation

In qualità di amministratore del cloud, è possibile aggiungere un elemento catalogo per il provisioning dei database di VMware Data Services Manager in Automation Service Broker di VMware Aria Automation.

Connessione di VMware Aria Automation a un dominio del carico di lavoro per VMware Private AI Foundation with NVIDIA

Prima di poter aggiungere elementi del catalogo per il provisioning delle applicazioni di intelligenza artificiale utilizzando VMware Aria Automation, connettere VMware Aria Automation a VMware Cloud Foundation.

Procedura

- Nell'interfaccia utente di VMware Aria Automation, eseguire la procedura guidata Avvio rapido per VMware Cloud Foundation o per VMware vCenter Server.

  Vedere Come iniziare a utilizzare VMware Aria Automation tramite Avvio rapido di VMware Cloud Foundation o Come iniziare a utilizzare VMware Aria Automation tramite Avvio rapido di VMware vCenter Server nella documentazione *Guida introduttiva a VMware Aria Automation*.

Creazione di elementi catalogo self-service AI in VMware Aria Automation

In qualità di amministratore del cloud, utilizzare la procedura guidata di configurazione del catalogo per Private AI in VMware Aria Automation per aggiungere rapidamente elementi catalogo per la distribuzione di istanze di Deep Learning VM o cluster TKG con accelerazione GPU in un dominio del carico di lavoro VI in VMware Cloud Foundation connesso.

I data scientist possono utilizzare gli elementi catalogo di deep learning per la distribuzione delle macchine virtuali di deep learning. I tecnici DevOps possono utilizzare gli elementi catalogo per il provisioning di cluster TKG pronti per AI.

Ogni volta che viene eseguita, la configurazione guidata del catalogo per Private AI aggiunge elementi per le istanze di Deep Learning VM e i cluster TKG nel catalogo di Service Broker. È possibile eseguire la procedura guidata ogni volta che è necessario:

- Abilitare il provisioning dei carichi di lavoro AI in un altro supervisore.
- Apportare una modifica alla licenza NVIDIA AI Enterprise, che include il file `.tok` per la configurazione del client e il server delle licenze o l'URL di download per i driver guest della vGPU per un ambiente disconnesso.
- Apportare una modifica all'immagine di una macchina virtuale di deep learning.
- Utilizzare altre classi di macchine virtuali vGPU o non GPU, un criterio di storage o un registro di container.

- Creare elementi catalogo in un nuovo progetto.

Nota VMware Aria Automation crea uno spazio dei nomi vSphere ogni volta che viene eseguito il provisioning di un'istanza di Deep Learning VM o di un cluster Tanzu Kubernetes Grid.

Procedura

- Aggiunta di elementi di Private AI al catalogo Automation Service Broker.

Operazioni successive

Utilizzando Automation Service Broker, i data scientist possono procedere con la distribuzione delle istanze di Deep Learning VM e i tecnici DevOps possono procedere con il provisioning di cluster Tanzu Kubernetes Grid abilitati per GPU. Vedere Distribuzione di Deep Learning VM non RAG in VMware Aria Automation.

Creazione di un elemento catalogo di un database vettore in VMware Aria Automation

In qualità di amministratore del cloud, è possibile aggiungere un elemento catalogo per il provisioning dei database di VMware Data Services Manager in Automation Service Broker di VMware Aria Automation.

Prerequisiti

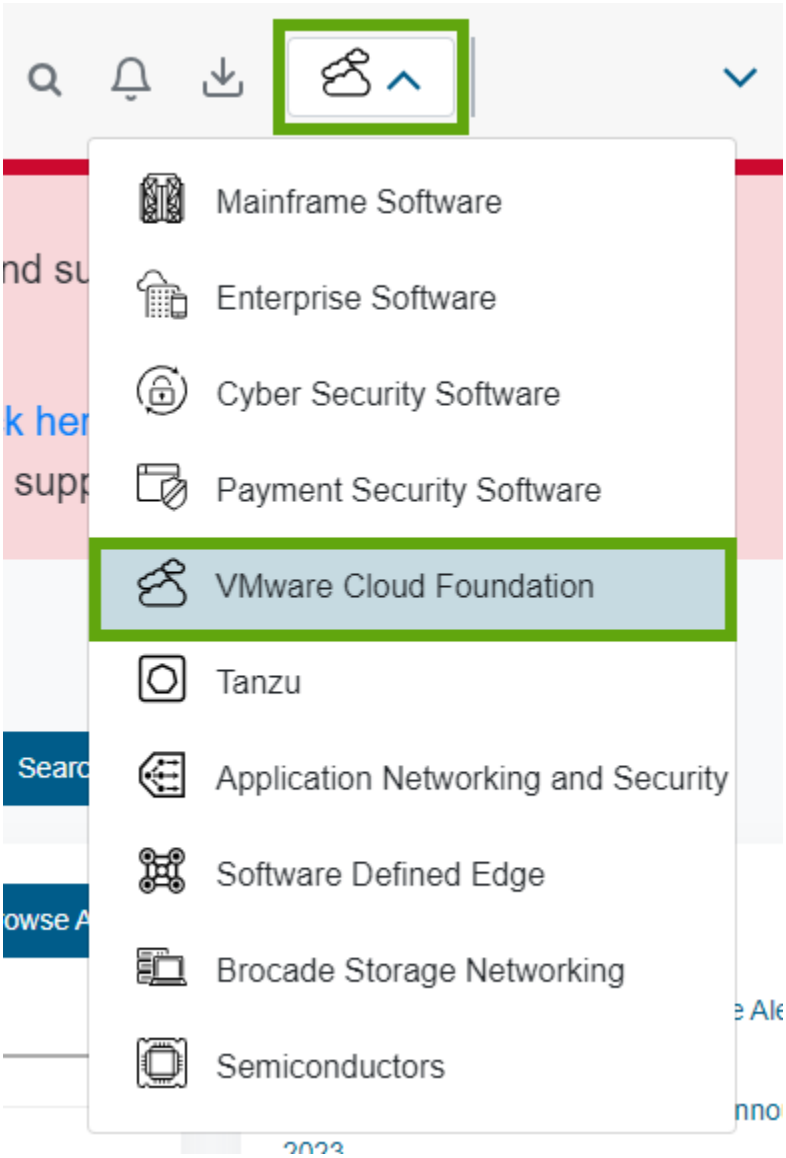
- Verificare che sia distribuito VMware Data Services Manager 2.1.
- Specificare una macchina in cui sia installato Python 3.10 che abbia accesso alle istanze di VMware Data Services Manager e VMware Aria Automation.

Procedura

1 Scaricare il bundle `AriaAutomation_DataServicesManager` per VMware Data Services Manager 2.1 dal portale tecnico di Broadcom.

   a Accedere al portale di supporto di Broadcom.

   b Dal menu a discesa della categoria del software nell'angolo in alto a destra del portale, selezionare **VMware Cloud Foundation**.



   c Nel riquadro di navigazione a sinistra fare clic su **My Downloads**.

   d Nella pagina **My Downloads - VMware Cloud Foundation** fare clic su **VMware Data Services Manager**.

   e Fare clic sul numero di versione e scaricare il bundle `AriaAutomation_DataServicesManager`.

2 Nella macchina che esegue Python, caricare il bundle `AriaAutomation_DataServicesManager` ed estrarne il contenuto.

3 Aggiornare il file `config.json` nella cartella in cui è stato estratto il bundle con gli URL e le credenziali utente per VMware Data Services Manager e VMware Aria Automation.

   Facoltativamente, è anche possibile impostare il nome dell'elemento catalogo, il progetto di Automation Assembler e altri parametri.

4 Per creare gli elementi catalogo in VMware Aria Automation, eseguire lo script Python `aria.py` nel modo seguente.

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

Risultati

Lo script Python crea in VMware Aria Automation elementi necessari per l'utilizzo di VMware Data Services Manager per il provisioning del database. Vedere il file `readme.md` nel bundle `AriaAutomation_DataServicesManager`

Operazioni successive

I data scientist o i tecnici DevOps possono distribuire un database vettore dal catalogo di Automation Service Broker con estensione pgvector e integrarlo nei carichi di lavoro RAG. Vedere Capitolo 5 Distribuzione di carichi di lavoro RAG in VMware Private AI Foundation with NVIDIA.

Distribuzione di una macchina virtuale di deep learning in VMware Private AI Foundation with NVIDIA

3

VMware Private AI Foundation with NVIDIA supporta il provisioning di istanze di Deep Learning VM preconfigurate che i data scientist possono utilizzare per lo sviluppo di AI.

Per un data scientist, sono disponibili le opzioni seguenti per iniziare a utilizzare Deep Learning VM:

- Distribuire Deep Learning VM mediante un elemento catalogo self-service in VMware Aria Automation.
- Richiedere al tecnico DevOps di distribuire Deep Learning VM in un cluster Tanzu Kubernetes Grid utilizzando il comando `kubectl`.
- Richiedere all'amministratore del cloud di distribuire Deep Learning VM in un cluster vSphere per esplorare rapidamente i modelli di Deep Learning VM.
- Informazioni sulle immagini delle macchine virtuali di deep learning in VMware Private AI Foundation with NVIDIA

  Le immagini delle macchine virtuali di deep learning fornite con VMware Private AI Foundation with NVIDIA sono preconfigurate con le librerie, i framework e i toolkit ML più comuni e sono ottimizzate e convalidate da NVIDIA e VMware per l'accelerazione della GPU in un ambiente VMware Cloud Foundation.
- Distribuzione di una macchina virtuale di deep learning tramite un catalogo self-service in VMware Aria Automation

  In VMware Private AI Foundation with NVIDIA in qualità di data scientist o tecnico DevOps, è possibile distribuire una macchina virtuale di deep learning da VMware Aria Automation utilizzando gli elementi catalogo self-service di una workstation AI in Automation Service Broker.
- Distribuzione di una macchina virtuale di deep learning direttamente in un cluster vSphere in VMware Private AI Foundation with NVIDIA

  Per dare rapidamente ai data scientist l'opportunità di testare i modelli di Deep Learning VM in VMware Private AI Foundation with NVIDIA, in qualità di amministratore del cloud è possibile distribuire Deep Learning VM direttamente in un cluster vSphere utilizzando vSphere Client.

- Distribuzione di una macchina virtuale di deep learning tramite il comando `kubectl` in VMware Private AI Foundation with NVIDIA

  Il servizio macchina virtuale nel supervisore in vSphere IaaS Control Plane consente ai tecnici DevOps di distribuire ed eseguire istanze di Deep Learning VM utilizzando l'API Kubernetes.

- Personalizzazione della distribuzione di Deep Learning VM in VMware Private AI Foundation with NVIDIA

  Quando si distribuisce un'istanza di Deep Learning VM in vSphere IaaS control plane utilizzando `kubectl` o direttamente in un cluster vSphere, è necessario compilare le proprietà personalizzate della macchina virtuale.

- Risoluzione dei problemi relativi alla distribuzione di un'istanza di Deep Learning VM in VMware Private AI Foundation with NVIDIA

  Le informazioni sulla risoluzione dei problemi relativi alla distribuzione di Deep Learning VM in VMware Private AI Foundation with NVIDIA forniscono soluzioni ai potenziali problemi che potrebbero verificarsi.

Informazioni sulle immagini delle macchine virtuali di deep learning in VMware Private AI Foundation with NVIDIA

Le immagini delle macchine virtuali di deep learning fornite con VMware Private AI Foundation with NVIDIA sono preconfigurate con le librerie, i framework e i toolkit ML più comuni e sono ottimizzate e convalidate da NVIDIA e VMware per l'accelerazione della GPU in un ambiente VMware Cloud Foundation.

In qualità di data scientist, è possibile utilizzare le istanze di Deep Learning VM fornite da queste immagini per la prototipazione, l'ottimizzazione, la convalida e l'inferenza di AI.

Lo stack software per l'esecuzione di applicazioni AI oltre alle GPU NVIDIA viene convalidato in anticipo. Di conseguenza, è possibile avviare direttamente lo sviluppo di AI senza perdere tempo per l'installazione e la convalida della compatibilità di sistemi operativi, librerie software, framework ML, toolkit e driver GPU.

Che cosa contiene un'immagine della macchina virtuale di deep learning?

L'immagine di Deep Learning VM più recente contiene il software seguente. Per informazioni sulle versioni dei componenti in ogni versione dell'immagine della macchina virtuale di deep learning, vedere Note di rilascio di VMware Deep Learning VM.

Categoria del componente software	Componente software		
Incorporato	■ Canonical Ubuntu ■ NVIDIA Container Toolkit ■ Docker Community Engine ■ Miniconda e un manifesto di PyTorch Conda.		
Può essere preinstallata automaticamente quando si avvia Deep Learning VM per la prima volta	■ Driver guest della vGPU in base alla versione del driver host della vGPU		
	Carichi di lavoro di Deep Learning (DL)	Esempio di CUDA È possibile utilizzare una macchina virtuale di deep learning con esempi CUDA in esecuzione per esplorare l'aggiunta di un vettore, la simulazione gravitazionale di n-corpi o altri esempi in una macchina virtuale. Vedere la pagina Esempi di CUDA nel catalogo NVIDIA NGC.	
		PyTorch. È possibile utilizzare una macchina virtuale di deep learning con una libreria PyTorch per esplorare AI conversazionale, l'elaborazione del linguaggio naturale (NLP) e altri tipi di modelli AI in una macchina virtuale. Vedere la pagina PyTorch nel catalogo NVIDIA NGC. È possibile utilizzare un'istanza di JupyterLab pronta con PyTorch installato e configurato all'indirizzo `http://dl_vm_ip:8888`.	
		TensorFlow. È possibile utilizzare una macchina virtuale di deep learning con una libreria TensorFlow per esplorare AI conversazionale, l'elaborazione del linguaggio naturale (NLP) e altri tipi di modelli AI in una macchina virtuale. Vedere la pagina TensorFlow nel catalogo NVIDIA NGC. È possibile utilizzare un'istanza di JupyterLab pronta con TensorFlow installato e configurato all'indirizzo `http://dl_vm_ip:8888`.	
		DCGM Exporter È possibile utilizzare un'istanza di Deep Learning VM con Data Center GPU Manager (DCGM) Exporter per monitorare l'integrità e ottenere le metriche delle GPU utilizzate da un carico di lavoro DL, tramite NVIDIA DCGM, Prometheus e Grafana. Vedere la pagina DCGM Exporter nel catalogo NVIDIA NGC. In una macchina virtuale di deep learning eseguire il container DCGM Exporter insieme a un carico di lavoro DL che esegue le operazioni di AI. Dopo l'avvio di Deep Learning VM, DCGM Exporter è pronto a raccogliere le metriche di vGPU ed esportare i dati in un'altra applicazione per ulteriore monitoraggio e visualizzazione. Per informazioni su come utilizzare DGCM Exporter per visualizzare le metriche con Prometheus e Grafana, vedere DCGM Exporter.	

Categoria del componente software	Componente software
	Triton Inference Server È possibile utilizzare un'istanza di Deep Learning VM con Triton Inference Server per caricare un repository di modelli e ricevere richieste di inferenza. Vedere la pagina Triton Inference Server nel catalogo NVIDIA NGC. Per informazioni su come utilizzare Triton Inference Server per le richieste di inferenza per i modelli di AI, vedere Triton Inference Server.
	NVIDIA RAG È possibile utilizzare un'istanza di Deep Learning VM per creare soluzioni RAG (Retrieval Augmented Generation) con un modello Llama2. Vedere la documentazione NVIDIA RAG Applications Docker Compose (richiede autorizzazioni specifiche dell'account). Un'applicazione Web chatbot di esempio a cui è possibile accedere all'indirizzo `http://dl_vm_ip:3001/orgs/nvidia/models/text-qa-chatbot`. È possibile caricare la propria knowledge base.

Distribuzione di Deep Learning VM

In qualità di data scientist, è possibile distribuire autonomamente Deep Learning VM utilizzando gli elementi del catalogo in VMware Aria Automation. Oppure, è possibile chiedere a un amministratore del cloud o un tecnico DevOps di distribuire tale istanza di VM.

Distribuzione di una macchina virtuale di deep learning tramite un catalogo self-service in VMware Aria Automation

In VMware Private AI Foundation with NVIDIA in qualità di data scientist o tecnico DevOps, è possibile distribuire una macchina virtuale di deep learning da VMware Aria Automation utilizzando gli elementi catalogo self-service di una workstation AI in Automation Service Broker.

Per informazioni sulle immagini di Deep Learning VM in VMware Private AI Foundation with NVIDIA, vedere Informazioni sulle immagini delle macchine virtuali di deep learning in VMware Private AI Foundation with NVIDIA.

Prerequisiti

- Verificare che l'amministratore del cloud abbia configurato il catalogo di VMware Aria Automation per la distribuzione dell'applicazione Private AI. Vedere Aggiunta di elementi di Private AI al catalogo Automation Service Broker.
- Verificare che l'amministratore del cloud abbia assegnato il ruolo utente necessario per la distribuzione delle istanze di Deep Learning VM.

Procedura

- Distribuzione di Deep Learning VM non RAG in VMware Aria Automation o Distribuzione di una macchina virtuale di deep learning con un carico di lavoro RAG.

  La distribuzione di un'istanza di Deep Learning VM con NVIDIA RAG richiede un database vettore, ad esempio un database PostgreSQL con pgvector in VMware Data Services Manager.

Risultati

Il driver guest della vGPU e il carico di lavoro di Deep Learning specificato vengono installati la prima volta che si avvia Deep Learning VM.

Operazioni successive

Per informazioni dettagliate su come accedere alla macchina virtuale e all'istanza di JupyterLab inclusa in alcune immagini di Deep Learning VM, in Automation Service Broker passare a **Utilizza > Distribuzioni > Distribuzioni**.

Distribuzione di una macchina virtuale di deep learning direttamente in un cluster vSphere in VMware Private AI Foundation with NVIDIA

Per dare rapidamente ai data scientist l'opportunità di testare i modelli di Deep Learning VM in VMware Private AI Foundation with NVIDIA, in qualità di amministratore del cloud è possibile distribuire Deep Learning VM direttamente in un cluster vSphere utilizzando vSphere Client.

Per informazioni sulle immagini di Deep Learning VM in VMware Private AI Foundation with NVIDIA, vedere Informazioni sulle immagini delle macchine virtuali di deep learning in VMware Private AI Foundation with NVIDIA.

La distribuzione di un'istanza di Deep Learning VM con NVIDIA RAG richiede un database vettore, ad esempio un database PostgreSQL con pgvector in VMware Data Services Manager. Per informazioni sulla distribuzione di tale database e sulla sua integrazione in Deep Learning VM, vedere Distribuzione di una macchina virtuale di deep learning con un carico di lavoro RAG.

Prerequisiti

Verificare che VMware Private AI Foundation with NVIDIA sia distribuito e configurato. Vedere Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI.

Procedura

1. Accedere all'istanza di vCenter Server per il dominio del carico di lavoro VI.
2. Dal menu Home di vSphere Client selezionare **Librerie di contenuti**.
3. Passare all'immagine della macchina virtuale di deep learning nella libreria di contenuti.

4 Fare clic con il pulsante destro del mouse su un modello OVF e scegliere **Nuova macchina virtuale da questo modello**.

5 Nella pagina **Seleziona nome e cartella** della procedura guidata visualizzata, immettere un nome e selezionare una cartella della macchina virtuale, scegliere **Personalizza hardware di questa macchina virtuale** e fare clic su **Avanti**.

6 Selezionare un cluster abilitato per GPU nel dominio del carico di lavoro VI, scegliere se la macchina virtuale deve essere accesa al termine della distribuzione e fare clic su **Avanti**.

7 Seguire le indicazioni della procedura guidata per selezionare un datastore e una rete nel Distributed Switch per il cluster.

8 Nella pagina **Personalizza modello** immettere le proprietà della macchina virtuale personalizzata necessarie per configurare la funzionalità AI e fare clic su **Avanti**.

   Vedere Proprietà OVF delle macchine virtuali di deep learning.

9 Nella pagina **Personalizza hardware** assegnare un dispositivo NVIDIA vGPU alla macchina virtuale come **Nuovo dispositivo PCI** e fare clic su **Avanti**.

   Per una macchina virtuale di deep learning che esegue NVIDIA RAG, selezionare il profilo vGPU completo per la modalità di slicing temporale o un profilo MIG. Ad esempio, per NVIDIA A100 a 40 GB in modalità di slicing temporale vGPU, selezionare **nvidia_a100-40c**.

10 Per una macchina virtuale di deep learning che esegue NVIDIA RAG, nella scheda **Parametri avanzati** delle impostazioni della macchina virtuale, impostare il parametro `pciPassthru<vgpu-id>.cfg.enable_uvm` su **1**.

   dove `<vgpu-id>` identifica la vGPU assegnata alla macchina virtuale. Ad esempio, se alla macchina virtuale sono assegnate due vGPU, impostare `pciPassthru0.cfg.parameter=1` e `pciPassthru1.cfg.parameter = 1`.

11 Rivedere la specifica della distribuzione e fare clic su **Fine**.

Risultati

Il driver guest della vGPU e il carico di lavoro di deep learning specificato vengono installati la prima volta che si avvia la macchina virtuale di deep learning.

È possibile esaminare i registri o aprire l'istanza di JupyterLab fornita con alcune delle immagini. È possibile condividere i dettagli di accesso con i data scientist dell'organizzazione. Vedere Carichi di lavoro di deep learning in VMware Private AI Foundation with NVIDIA.

Operazioni successive

- Connettersi alla macchina virtuale di deep learning tramite SSH e verificare che tutti i componenti siano installati e in esecuzione come previsto.
- Inviare i dettagli di accesso ai data scientist.

Distribuzione di una macchina virtuale di deep learning tramite il comando `kubectl` in VMware Private AI Foundation with NVIDIA

Il servizio macchina virtuale nel supervisore in vSphere IaaS Control Plane consente ai tecnici DevOps di distribuire ed eseguire istanze di Deep Learning VM utilizzando l'API Kubernetes.

In qualità di tecnico DevOps, utilizzare `kubectl` per distribuire un'istanza di Deep Learning VM nello spazio dei nomi configurato dall'amministratore del cloud.

Per informazioni sulle immagini di Deep Learning VM in VMware Private AI Foundation with NVIDIA, vedere Informazioni sulle immagini delle macchine virtuali di deep learning in VMware Private AI Foundation with NVIDIA.

La distribuzione di un'istanza di Deep Learning VM con NVIDIA RAG richiede un database vettore, ad esempio un database PostgreSQL con pgvector in VMware Data Services Manager. Per informazioni sulla distribuzione di tale database e sulla sua integrazione in Deep Learning VM, vedere Distribuzione di una macchina virtuale di deep learning con un carico di lavoro RAG.

Prerequisiti

Verificare con l'amministratore del cloud che VMware Private AI Foundation with NVIDIA sia distribuito e configurato. Vedere Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI.

Procedura

1. Accedere al piano di controllo del supervisore.

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

2. Verificare che tutte le risorse macchina virtuale necessarie, ad esempio le classi di macchine virtuali e le immagini di macchine virtuali, siano presenti nello spazio dei nomi.

   Vedere Visualizzazione delle risorse macchina virtuale disponibili in uno spazio dei nomi in vSphere with Tanzu.

3. Preparare il file YAML per la macchina virtuale di deep learning.

   Utilizzare vm-operator-api, impostando le proprietà OVF come oggetto ConfigMap. Per informazioni sulle proprietà OVF disponibili, vedere Proprietà OVF delle macchine virtuali di deep learning.

   Ad esempio, è possibile creare una specifica YAML `example-dl-vm.yaml` per un'istanza di Deep Learning VM di esempio che esegue PyTorch in un ambiente connesso.

   ```
   apiVersion: vmoperator.vmware.com/v1alpha1
   kind: VirtualMachine
   metadata:
     name: example-dl-vm
   ```

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

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

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBwLnNoCiAgcGVybWlzc2lvbnM
6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbH
ZtL3V0aWxzLnNoCiAgICB0cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jrb
G9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCgogICAgREVGQVVMVF9SRUdfVVJJ
PSJudmNyLmlvIgogICAgUkVHSVNUUllfVVJJX1BBVEg9JChncmVwIHJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWV
udi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCgogICAgaWYgW1sgLXogIiRSRU
dJU1RSWV9VUklfUEFUSCIgXV07IHRoZW4KICAgICAgIyBJZiBSRUdJU1RSWV9VUklfUEFUSCBpcyBudWxsIG9yIGVtc
HR5LCB1c2UgdGhlIGRlZmF1bHQgdmFsdWUKICAgICAgUkVHSVNUUllfVVJJX1BBVEg9JERFRkFVTFRfUkVHX1VSSQog
ICAgICBlY2hvICJSRUdJU1RSWV9VUklfUEFUSCB3YXMgZW1wdHkuIFVzaW5nIGRlZmF1bHQ6ICRSRUdJU1RSWV9VUkl
fUEFUSCIKICAgIGZpCiAgICAKICAgICMgSWYgUkVHSVNUUllfVVJJX1BBVEggY29udGFpbnMgJy8nLCBleHRyYWN0IH
RoZSBVUkkgcGFydAogICAgaWYgW1sgJFJFR0lTVFJZX1VSSV9QQVRIID09ICoiLyIqIF1dOyB0aGVuCiAgICAgIFJFR
0lTVFJZX1VSST0kKGVjaG8gIiRSRUdJU1RSWV9VUklfUEFUSCIgfCBjdXQgLWQnLycgLWYxKQogICAgZWxzZQogICAg
ICBSRUdJU1RSWV9VUkk9JFJFR0lTVFJZX1VSSV9QQVRICiAgICBmaQogIAogICAgUkVHSVNUUllfVVNFUk5BTUU9JCh
ncmVwIHJlZ2lzdHJ5LXVzZXIgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW1
4iXSpcKS4qL1wxL3AnKQogICAgUkVHSVNUUllfUEFTU1dPUkQ9JChncmVwIHJlZ2lzdHJ5LXBhc3N3ZCAvb3B0L2Rsd
m0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCiAgICBpZiBbWyAt
biAiJFJFR0lTVFJZX1VTRVJOQU1FIiAmJiAtbiAiJFJFR0lTVFJZX1BBU1NXT1JEIiBdXTsgdGhlbgogICAgICBkb2N
rZXIgbG9naW4gLXUgJFJFR0lTVFJZX1VTRVJOQU1FIC1wICRSRUdJU1RSWV9QQVNTV09SRCAkUkVHSVNUUllfVVJJCi
AgICBlbHNlCiAgICAgIGVjaG8gIldhcm5pbmc6IHRoZSByZWdpc3RyeSdzIHVzZXJuYW1lIGFuZCBwYXNzd29yZCBhc
mUgaW52YWxpZCwgU2tpcHBpbmcgRG9ja2VyIGxvZ2luLiIKICAgIGZpCgogICAgZG9ja2VyIHJ1biAtZCAtLWdwdXMg
YWxsIC1wIDg4ODg6ODg4OCAkUkVHSVNUUllfVVJJX1BBVEgvbnZpZGlhL3B5dG9yY2g6MjMuMTAtcHkzIC91c3IvbG9
jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9vdCAtLWlwPSogLS1wb3J0PTg4ODggLS1uby1icm93c2VyIC0tTm
90ZWJvb2tBcHAudG9rZW49JycgLS1Ob3RlYm9va0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGlyPS93b
3Jrc3BhY2UKCi0gcGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6
IHwKICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAgICBlY2hvICJFcnJvcjogJDEiID4mMgogICA
gICB2bXRvb2xzZCAtLWNtZCAiaW5mby1zZXQgZ3Vlc3RpbmZvLnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIG
ZhbHNlLCBETFdvcmtsb2FkRmFpbHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgfQoKICAgIGNoZWNrX3Byb3RvY29sK
CkgewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQKICAgICAgbG9jYWwgc3VwcG9ydGVkX3Byb3Rv
Y29scz0oIiRAIikKICAgICAgaWYgW1sgLW4gIiR7cHJveHlfdXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm9
0b2NvbD0kKGVjaG8gIiR7cHJveHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBwcmludCAkMTsgZW
xzZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2NvbCIgXTsgdGhlbgogICAgICAgICAgZWNobyAiT
m8gc3BlY2lmaWMgcHJvdG9jb2wgcHJvdmlkZWQuIFNraXBwaW5nIHByb3RvY29sIGNoZWNrLiIKICAgICAgICAgIHJl
dHVybiAwCiAgICAgICAgZmkKICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQogICAgICAgIGZvciB
2YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwogICAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2x9Ii
A9PSAiJHt2YXJ9IiBdXTsgdGhlbgogICAgICAgICAgICBwcm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgI
GJyZWFrCiAgICAgICAgICBmaQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2NvbF9pbmNsdWRlZH0i
```

```
ID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0ICJVbnN1cHBvcnRlZCBwcm90b2NvbDogJHtwcm9
0b2NvbH0uIFN1cHBvcnRlZCBwcm90b2NvbHMgYXJlOiAke3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgIG
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
```

Nota `user-data` è il valore codificato in base64 per il codice cloud-init seguente:

```
#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 Passare al contesto dello spazio dei nomi vSphere creato dall'amministratore del cloud.

Ad esempio, per uno spazio dei nomi denominato `example-dl-vm-namespace`:

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

5 Distribuire la macchina virtuale di deep learning.

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

6 Verificare che la macchina virtuale sia stata creata eseguendo questi comandi.

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

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

7 Eseguire il ping dell'indirizzo IP della macchina virtuale assegnato dal servizio di rete richiesto.

Per ottenere l'indirizzo pubblico e le porte per l'accesso alla macchina virtuale di deep learning, recuperare i dettagli relativi al servizio di bilanciamento del carico creato.

```
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
```

Risultati

Il driver guest della vGPU e il carico di lavoro DL specificato vengono installati la prima volta che si avvia la macchina virtuale di deep learning.

Operazioni successive

- È possibile esaminare i registri o aprire il notebook JupyterLab fornito con alcune delle immagini. Vedere Carichi di lavoro di deep learning in VMware Private AI Foundation with NVIDIA.
- Inviare i dettagli di accesso ai data scientist.

Personalizzazione della distribuzione di Deep Learning VM in VMware Private AI Foundation with NVIDIA

Quando si distribuisce un'istanza di Deep Learning VM in vSphere IaaS control plane utilizzando `kubectl` o direttamente in un cluster vSphere, è necessario compilare le proprietà personalizzate della macchina virtuale.

Per informazioni sulle immagini di Deep Learning VM in VMware Private AI Foundation with NVIDIA, vedere Informazioni sulle immagini delle macchine virtuali di deep learning in VMware Private AI Foundation with NVIDIA.

Proprietà OVF delle macchine virtuali di deep learning

Quando si distribuisce un'istanza di Deep Learning VM, è necessario compilare le proprietà della macchina virtuale personalizzate per automatizzare la configurazione del sistema operativo Linux, la distribuzione del driver guest della vGPU, nonché la distribuzione e la configurazione dei container NGC per i carichi di lavoro DL.

L'immagine di Deep Learning VM più recente ha le proprietà OVF seguenti:

Categoria	Parametro	Etichetta in vSphere Client	Descrizione
Proprietà del sistema operativo di base	instance-id	ID istanza	Obbligatorio. ID istanza univoco per la macchina virtuale. Un ID istanza identifica in modo univoco un'istanza. Quando un ID istanza viene modificato, cloud-init gestisce l'istanza come una nuova istanza ed esegue nuovamente il processo cloud-init.
	hostname	Nome host	Obbligatorio. Nome host dell'appliance.
	seedfrom	URL da cui effettuare il seeding dei dati dell'istanza	Facoltativo. URL da cui estrarre il valore del parametro user-data e dei metadati.
	public-keys	Chiave pubblica SSH	Se specificato, l'istanza popola il valore `authorized_keys` di SSH dell'utente predefinito con questo valore.

Categoria	Parametro	Etichetta in vSphere Client	Descrizione
	user-data	Dati utente codificati	Un set di script o altri metadati che viene inserito nella macchina virtuale al momento del provisioning. Questa proprietà rappresenta il contenuto effettivo dello script `cloud-init`. Questo valore deve essere codificato tramite base64. ■ È possibile utilizzare questa proprietà per specificare il container del carico di lavoro DL che si desidera distribuire, ad esempio PyTorch o TensorFlow. Vedere Carichi di lavoro di deep learning in VMware Private AI Foundation with NVIDIA. ■ Questa proprietà consente di impostare un indirizzo IP statico per una macchina virtuale distribuita direttamente in un cluster vSphere. Vedere Assegnazione di un indirizzo IP statico a una macchina virtuale di deep learning in VMware Private AI Foundation with NVIDIA.
	password	Password utente predefinito	Obbligatorio. Password dell'account utente **vmware** predefinito.
Installazione del driver vGPU	vgpu-license	Licenza vGPU	Obbligatorio. Token di configurazione del client NVIDIA vGPU. Il token viene salvato nel file `/etc/nvidia/ClientConfigToken/client_configuration_token.tok`.
	nvidia-portal-api-key	Chiave API portale NVIDIA	Obbligatorio in un ambiente connesso. Chiave API scaricata dal portale delle licenze NVIDIA. La chiave è necessaria per l'installazione del driver guest della vGPU.
	vgpu-fallback-version	Versione driver host vGPU	Installa direttamente questa versione del driver guest della vGPU.
	vgpu-url	URL per i download della vGPU air gap	Obbligatorio in un ambiente disconnesso. URL da cui scaricare il driver guest della vGPU. Per informazioni sulla configurazione necessaria del server Web locale, vedere Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI.
Automazione del carico di lavoro DL	registry-uri	URI registro	Obbligatorio in un ambiente disconnesso o se si intende utilizzare un registro di container privato per evitare di scaricare immagini da Internet. URI di un registro di container privato con le immagini del container del carico di lavoro di deep learning. Obbligatorio se si fa riferimento a un registro privato in `user-data` o `image-oneliner`.
	registry-user	Nome utente registro	Obbligatorio se si utilizza un registro di container privato che richiede l'autenticazione di base.
	registry-passwd	Password registro	Obbligatorio se si utilizza un registro di container privato che richiede l'autenticazione di base.

Categoria	Parametro	Etichetta in vSphere Client	Descrizione
	registry-2-uri	URI registro secondario	Obbligatorio se si utilizza un secondo registro di container privato basato su Docker che richiede l'autenticazione di base. Ad esempio, quando si distribuisce un'istanza di Deep Learning VM con il carico di lavoro DL di NVIDIA RAG preinstallato, un'immagine di pgvector viene scaricata da Docker Hub. È possibile utilizzare i parametri `registry-2-` per ignorare un limite di velocità pull per docker.io.
	registry-2-user	Nome utente registro secondario	Obbligatorio se si utilizza un secondo registro di container privato.
	registry-2-passwd	Password registro secondario	Obbligatorio se si utilizza un secondo registro di container privato.
	image-oneliner	Comando a una riga codificato	Comando bash a una riga che viene eseguito al momento del provisioning della macchina virtuale. Questo valore deve essere codificato tramite base64. È possibile utilizzare questa proprietà per specificare il container del carico di lavoro DL che si desidera distribuire, ad esempio PyTorch o TensorFlow. Vedere Carichi di lavoro di deep learning in VMware Private AI Foundation with NVIDIA. **Attenzione** Evitare di utilizzare sia `user-data` sia `image-oneliner`.
	docker-compose-uri	File di composizione Docker codificato	Obbligatorio se è necessario un file di composizione di Docker per avviare il container di carichi di lavoro DL. Contenuti del file `docker-compose.yaml` che verranno inseriti nella macchina virtuale al momento del provisioning dopo l'avvio della macchina virtuale con la GPU abilitata. Questo valore deve essere codificato tramite base64.
	config-json	config.json codificato	Contenuti di un file di configurazione per l'aggiunta di dettagli per i server proxy. Questo valore deve essere codificato tramite base64. Vedere Configurazione di un'istanza di Deep Learning VM con un server proxy.
	conda-environment-install	Installazione dell'ambiente Conda	Elenco di ambienti Conda separati da virgole da installare automaticamente al termine della distribuzione della macchina virtuale. Ambienti disponibili: `pytorch2.3_py3.12`

Carichi di lavoro di deep learning in VMware Private AI Foundation with NVIDIA

È possibile eseguire il provisioning di una macchina virtuale di deep learning con un carico di lavoro di deep learning (DL) supportato oltre ai suoi componenti incorporati. I carichi di lavoro DL

vengono scaricati dal catalogo NVIDIA NGC e sono ottimizzati per la GPU e convalidati da NVIDIA e VMware by Broadcom.

Per una panoramica delle immagini delle macchine virtuali di deep learning, vedere Informazioni sulle immagini delle macchine virtuali di deep learning in VMware Private AI Foundation with NVIDIA.

Esempio di CUDA

È possibile utilizzare una macchina virtuale di deep learning con esempi CUDA in esecuzione per esplorare l'aggiunta di un vettore, la simulazione gravitazionale di n-corpi o altri esempi in una macchina virtuale. Vedere la pagina Esempi di CUDA.

Dopo l'avvio, la macchina virtuale di deep learning esegue un carico di lavoro di esempio CUDA per testare il driver guest della vGPU. È possibile esaminare l'output del test nel file `/var/log/dl.log`.

Tabella 3-1. Immagine del container di esempio CUDA

Componente	Descrizione
Immagine del container	`nvcr.io/nvidia/k8s/cuda-sample:ngc_image_tag` Ad esempio: `nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8` Per informazioni sulle immagini di container di esempio CUDA supportate per le macchine virtuali di deep learning, vedere Note di rilascio di VMware Deep Learning VM.
Input necessari	Per distribuire un carico di lavoro di esempio CUDA, è necessario impostare le proprietà OVF per la macchina virtuale di deep learning nel modo seguente: ■ Utilizzare una delle proprietà seguenti specifiche per l'immagine dell'esempio CUDA. ■ Script cloud-init. Codificarlo nel formato 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
- path: /opt/dlvm/utils.sh
```

Tabella 3-1. Immagine del container di esempio CUDA (continua)

Componente	Descrizione

```
  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
```

Tabella 3-1. Immagine del container di esempio CUDA (continua)

Componente	Descrizione

```
        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"
    }
```

Ad esempio, per vectoradd-cuda11.7.1-ubi8, specificare lo script seguente in formato base64:

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICBz
ZXRfcHJveHkgImh0dHAiICJodHRwcyIgInNvY2tzNSIKICAgIHRyYXAgJ2Vycm9yX2V4
aXQgIlVuZXhwZWN0ZWQgZXJyb3Igb2NjdXJzIGF0IGRsIHdvcmtsb2FkIicgRVJSCiAg
ICBERUZBVUxUX1JFR19VUkk9Im52Y3IuaW8iCiAgICBSRUdJU1RSWV9VUklfUEFUSD0k
KGdyZXAgcmVnaXN0cnktdXJpIC9vcHQvZGx2bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAn
cy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKICAgICBpZiBbWyAteiAiJFJF
R0lTVFJZX1VSSV9QQVRIIiBdXTsgdGhlbgogICAgICAjIElmIFJFR0lTVFJZX1VSSV9Q
QVRIIGlzIG51bGwgb3IgZW1wdHksIHVzZSB0aGUgZGVmYXVsdCB2YWx1ZQogICAgICBS
RUdJU1RSWV9VUklfUEFUSD0kREVGQVVMVF9SRUdfVVJJCiAgICAgICAgIGVjaG8gIlJFR0lT
VFJZX1VSSV9QQVRIIHdhcyBlbXB0eS4gVXNpbmcgZGVmYXVsdDogJFJFR0lTVFJZX1VS
SV9QQVRIIgogICAgZmkKICAgIAogICAgIyBJZiBSRUdJU1RSWV9VUklfUEFUSCBjb250
YWluucyAnLycsIGV4dHJhY3QgdGhlIFVSSSBwYXJ0CiAgICBpZiBbWyAkUkVHSVNUUllf
VVJJX1BBVEggPT0gKiIvIiogXV07IHRoZW4KICAgICAgUkVHSVNUUllfVVJJPSQoZWNo
byAiJFJFR0lTVFJZX1VSSV9QQVRIIiB8IGN1dCAtZCcvJyAtZjEpCiAgICBlbHNlCiAg
ICAgIFJFR0lTVFJZX1VSST0kUkVHSVNUUllfVVJJX1BBVEgKICAgIGZpCiAgICBS
RUdJU1RSWV9VU0VSTkFNRT0kKGdyZXAgcmVnaXN0cnktdXNlciAvb3B0L2Rsdm0vb3Zm
LWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcp
CiAgICBSRUdJU1RSWV9QQVNTV09SRD0kKGdyZXAgcmVnaXN0cnktcGFzc3dkIC9vcHQv
ZGx2bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCku
Ki9cMS9wJykKICAgIGlmIFtbIC1uICIkUkVHSVNUUllfVVNFUk5BTUUiICYmIC1uICIk
UkVHSVNUUllfUEFTU1dPUkQiIF1dOyB0aGVuCiAgICAgIGRvY2tlciBsb2dpbiAtdSAk
UkVHSVNUUllfVVNFUk5BTUUgLXAgJFJFR0lTVFJZX1BBU1NXT1JEICRSRUdJU1RSWV9V
UkkKICAgIGVsc2UKICAgICAgZWNobyAiV2FybmluZzogdGhlIHJlZ2lzdHJ5J3MgdXNl
cm5hbWUgYW5kIHBhc3N3b3JkIGFyZSBpbnZhbGlkLCBtYSI2lwcGluZyBEb2NrZXIgbG9n
aW4uIgogICAgZmkKICAgIAogICAgZG9ja2VyIHJ1biAtZCAkUkVHSVNUUllfVVJJX1BB
VEgvbnZpZGlhL2s4cy9jdWRhLXNhbXBsZTp2ZWN0b3JhZGQtY3VkYTExLjcuMS11Ymk4
CgotIHBhdGg6IC9vcHQvZGx2bS91dGlscy5zaAogIHBlcm1pc3Npb25zOiAnMDc1NScK
ICBjb250ZW50OiB8CiAgICAgIS9iaW4vYmFzaAogICAgZXJyb3JfZXhpdCgpIHsKICAg
ICAgZWNobyAiRXJyb3I6ICQxIiA+JjIKICAgICAgdm10b29sc2QgLS1jbWQgImluZm8t
c2V0IGd1ZXN0aW5mby52bXNlcnZpY2UuYm9vdHN0cmFwLmNvbmRpdGlvbiBmYWxzZSwg
RExXb3JrbG9hZEZhaWx1cmUsICQxIgogICAgICBleGl0IDEKICAgICAgICAgICAgICAg
a19wcm90b2NvbCgpIHsKICAgICAgbG9jYWwgcHJveHlfdXJsPSQxCiAgICAgIHNoaWZ0
CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIpCiAgICAgIGlmIFtb
```

Tabella 3-1. Immagine del container di esempio CUDA (continua)

Componente	Descrizione
	IC1uICIke3Byb3h5X3VybH0iIF1dOyB0aGVuCiAgICAgICAgbG9jYWwgcHJvdG9jb2w9 JChlY2hvICIke3Byb3h5X3VybH0iIHwgYXdrIC1GICc6Ly8nICd7aWYgKE5GID4gMSkg cHJpbnQgJDE7IGVsc2UgcHJpbnQgIiJ9JykKICAgICAgICBpZiBbIC16ICIkcHJvdG9j b2wiIF07IHRoZW4KICAgICAgICAgICAgIGVjaG8gIk5vIHNwZWNpZmljIHByb3RvY29sIHBy b3ZpZGVkLiBTa2lwcGluZyBwcm90b2NvbCBjaGVjay4iCiAgICAgICAgICAgICBByZXR1cm4g MAogICAgICAgICAgIGZpCiAgICAgICAgbG9jYWwgcHJvdG9jb2xfaW5jbHVkZWQ9ZmFsc2UK ICAgICAgICAgICBmb3IgdmFyIGluICIke3N1cHBvcnRlZF9wcm90b2NvbHNbQF19IjsgZG8K ICAgICAgICAgICAgIGlmIFtbICIke3Byb3RvY29sfSIgPT0gIiR7dmFyfSIgXV07IHRoZW4K ICAgICAgICAgICAgICAgcHJvdG9jb2xfaW5jbHVkZWQ9dHJ1ZQogICAgICAgICAgICAgICAgICBicmVh awogICAgICAgICAgICAgZmkKICAgICAgICBkb25lCiAgICAgICAgaWYgW1sgIiR7cHJvdG9j b2xfaW5jbHVkZWR9IiA9PSBmYWxzZSBdXTsgdGhlbgogICAgICAgICAgICAgZXJyb3JfZXhp dCAiVW5zdXBwb3J0ZWQgcHJvdG9jb2w6ICR7cHJvdG9jb2x9LiBTdXBwb3J0ZWQgcHJv dG9jb2xzIGFyZTogJHtzdXBwb3J0ZWRfcHJvdG9jb2xzWypdfSIKICAgICAgICAgICAgICBmaQog ICAgICBmaQogICAgfQoKICAgICMgJEA6IGxpc3Qgb2Ygc3VwcG9ydGVkIHByb3RvY29s cwogICAgc2V0X3Byb3h5KCkgewogICAgICBsb2NhbCBzdXBwb3J0ZWRfcHJvdG9jb2xz PSgiJEAiKQoKICAgICAgQ09ORklHX0pTT05fQkFTRTY0PSQoZ3JlcCAnY29uZmlnLWpz b24nIC9vcHQvZGx2bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJc KFteIl0qXCkuKi9cMS9wJykKICAgICAgQ09ORklHX0pTT049JChlY2hvICR7Q09ORklH X0pTT05fQkFTRTY0fSB8IGJhc2U2NCAtLWRlY29kZSkKCiAgICAgICAgIEhUVFBfUFJPWFlf VVJMPSQoZWNobyAiJHtDT05GSUdfSlNPTn0iIHwganEgLXIgJy5odHRwX3Byb3h5IC8v IGVtcHR5JykKICAgICAgSFRUUFNfUFJPWFlfVVJMPSQoZWNobyAiJHtDT05GSUdfSlNP Tn0iIHwganEgLXIgJy5odHRwc19wcm94eSAvLyBlbXB0eScpCiAgICAgIGlmIFtbICQ/ IC1uZSAwIHx8ICgteiAiJHtIVFRQX1BST1hZX1VSTH0iICYmIC16ICIke0hUVFBTX1BS T1hZX1VSTH0iKSBdXTsgdGhlbgogICAgICAgICAgIGVjaG8gIkluZm86IFRoZSBjb25maWct anNvbiB3YXMgcGFyc2VkLCBidXQgbm8gcHJveHkgc2V0dGluZ3Mgd2VyZSBmb3VuZC4i CiAgICAgICAgcmV0dXJuIDAKICAgICAgZmkKICAKICAgICAgY2hlY2tfcHJvdG9jb2wg IiR7SFRUUF9QUk9YWV9VUkx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAg ICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVk X3Byb3RvY29sc1tAXX0iCgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9l dGMvZW52aXJvbm1lbnQ7IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94 eT0ke0hUVFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hU VFBTX1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJP WFlfVVJMfQogICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VS TH0KICAgICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4g L2V0Yy9lbnZpcm9ubWVudAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAg ICAgIGZpCiAgICAgIAogICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJv eHkKICAgICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2Vydmlj ZS5kCiAgICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBf UFJPWFk9JHtIVFRQX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBT X1BST1hZPSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25tZW50PVwiTk9f UFJPWFk9bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0v ZG9ja2VyLnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgIHN5c3RlbWN0bCBkYWVtb24t cmVsb2FkCiAgICAgIHN5c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAi SW5mbzogZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1 cmVkIHRvIHVzZSB0aGUgcHJveHkgc2V0dGluZ3MiCiAgICB9

che corrisponde allo script seguente in formato testo normale:

```
#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 |
```

Tabella 3-1. Immagine del container di esempio CUDA (continua)

Componente	Descrizione

```
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
            break
```

Tabella 3-1. Immagine del container di esempio CUDA (continua)

Componente	Descrizione

```
          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"
    }
```

- Immagine one-liner. Codificarlo nel formato base64

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

Tabella 3-1. Immagine del container di esempio CUDA (continua)

Componente	Descrizione
	Ad esempio, per vectoradd-cuda11.7.1-ubi8, specificare lo script seguente in formato base64: `ZG9ja2VyIHJ1biAtZCBudmNyLmlvL252aWRpYS9rOHMvY3VkYS1zYW1wbGU6dmVjdG9yYWRkLWN1ZGExMS43LjEtdWJpOA==` che corrisponde allo script seguente in formato testo normale: `docker run -d nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8` ■ Immettere le proprietà di installazione del driver guest vGPU, ad esempio `vgpu-license` e `nvidia-portal-api-key`. ■ Specificare i valori per le proprietà necessarie per un ambiente disconnesso in base alle esigenze. Vedere Proprietà OVF delle macchine virtuali di deep learning.
Output	■ Registri di installazione per il driver guest della vGPU in `/var/log/vgpu-install.log`. Per verificare che il driver guest della vGPU sia installato e che la licenza sia allocata, eseguire il comando seguente: `nvidia-smi -q |grep -i license` ■ Registri dello script cloud-init in `/var/log/dl.log`.

PyTorch

È possibile utilizzare una macchina virtuale di deep learning con una libreria PyTorch per esplorare AI conversazionale, l'elaborazione del linguaggio naturale (NLP) e altri tipi di modelli AI in una macchina virtuale. Vedere la pagina di PyTorch.

Dopo l'avvio, la macchina virtuale di deep learning avvia un'istanza di JupyterLab con i pacchetti PyTorch installati e configurati.

Tabella 3-2. Immagine del container PyTorch

Componente	Descrizione
Immagine del container	`nvcr.io/nvidia/pytorch:ngc_image_tag` Ad esempio: `nvcr.io/nvidia/pytorch:23.10-py3` Per informazioni sulle immagini dei container PyTorch supportate per le macchine virtuali di deep learning, vedere Note di rilascio di VMware Deep Learning VM.
Input necessari	Per distribuire un carico di lavoro PyTorch, è necessario impostare le proprietà OVF per la macchina virtuale di deep learning nel modo seguente: ■ Utilizzare una delle proprietà seguenti specifiche per l'immagine PyTorch. ■ Script cloud-init. Codificarlo nel formato 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/
nvidia/pytorch:ngc_image_tag /usr/local/bin/jupyter lab --allow-
```

Tabella 3-2. Immagine del container PyTorch (continua)

Componente	Descrizione

```
root --ip=* --port=8888 --no-browser --NotebookApp.token='' --
NotebookApp.allow_origin='*' --notebook-dir=/workspace

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

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

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

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

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

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

Tabella 3-2. Immagine del container PyTorch (continua)

Componente	Descrizione

```
      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"
    }
```

Ad esempio, per pytorch:23.10-py3, specificare lo script seguente in formato base 64:

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCgog
ICAgREVGQVVMVF9SRUdfVVJJPSJudmNyLmlvIgogICAgUkVHSVNUUllfVVJJX1BBVEg9
JChncmVwIHJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4g
J3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCgogICAgaWYgWlsgLXogIiRS
RUdJU1RSWV9VUklfUEFUSCIgXV07IHRoZW4KICAgICAgIyBJZiBSRUdJU1RSWV9VUklf
UEFUSCBpcyBudWxsIG9yIGVtcHR5LCB1c2UgdGhlIGRlZmF1bHQgdmFsdWUKICAgICAg
UkVHSVNUUllfVVJJX1BBVEg9JERERFRkFVTFRfUkVHX1VSSQogICAgICBlY2hvICJSRUdJ
U1RSWV9VUklfUEFUSCB3YXMgZW1wdHkuIFVzaW5nIGRlZmF1bHQ6ICRSRUdJU1RSWV9V
UklfUEFUSCIKICAgIGZpCiAgICAKICAgICMgSWYgUkVHSVNUUllfVVJJX1BBVEggY29u
dGFpbnMgJy8nLCBleHRyYWN0IHRoZSBVUkkgcGFydAogICAgaWYgWlsgJFJFR0lTVEFJZ
X1VSSV9QQVRIID09ICoiLyIqIF1dOyB0aGVuCiAgICAgIFJFR0lTVEFJZX1VSST0kKGVj
aG8gIiRSRUdJU1RSWV9VUklfUEFUSCIgfCBjdXQgLWQnLycgLWYxKQogICAgZWxzZQog
ICAgICBSRUdJU1RSWV9VUkk9JFJFR0lTVEFJZX1VSSV9QQVRICiAgICBmaQogICAgICAg
UkVHSVNUUllfVVNFUk5BTUU9JChncmVwIHJlZ2lzdHJ5LXVzZXIgL29wdC9kbHZtL292
Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3An
KQogICAgUkVHSVNUUllfUEFTU1dPUkQ9JChncmVwIHJlZ2lzdHJ5LXBhc3N3ZCAvb3B0
L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwp
LiovXDEvcCcpCiAgICBpZiBbWyAtbiAiJFJFR0lTVFJZX1VTRVJOQU1FIiAmJiAtbiAi
JFJFR0lTVFJZX1BBU1NXT1JEIiBdXTsgdGhlbgogICAgICBkb2NrZXIgbG9naW4gLXUg
JFJFR0lTVFJZX1VTRVJOQU1FIC1wICRSRUdJU1RSWV9QQVNTV09SRCAkUkVHSVNUUllf
VVJJCiAgICBlbHNlCiAgICAgIGVjaG8gIldhcm5pbmc6IHRoZSByZWdpc3RyeSHVz
ZXJuYW1lIGFuZCBwYXNzd29yZCBhcmUgaW52YWxpZCwgU2tpHBpbmcgRG9ja2VyIGxv
Z2luLiIKICAgIGZpCgogICAgZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC1wIDg4ODg6
ODg4OCAkUkVHSVNUUllfVVJJX1BBVEgvbnZpZGlhL3B5dG9yY2g6MjMuMTAtcHkzIC91
c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9vdCAtLW1wPSogLS1wb3J0
PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAudG9rZW49JycgLS1Ob3RlYm9v
a0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGlyPS93b3Jrc3BhY2UKCi0g
cGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNv
```

Tabella 3-2. Immagine del container PyTorch (continua)

Componente	Descrizione


```
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
```


che corrisponde allo script seguente in formato testo normale.

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

Tabella 3-2. Immagine del container PyTorch (continua)

Componente	Descrizione

```
    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 $1; else print ""}')
        if [ -z "$protocol" ]; then
```

Tabella 3-2. Immagine del container PyTorch (continua)

Componente	Descrizione

```
          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
```

Tabella 3-2. Immagine del container PyTorch (continua)

Componente	Descrizione
	<pre> echo "Info: docker and system environment are now configured to use the proxy settings" }</pre> ■ Immagine one-liner. Codificarlo nel formato 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> Ad esempio, per pytorch:23.10-py3, specificare lo script seguente in formato base 64: <pre>ZG9ja2VyIHJ1biAtZCAtcCA4ODg4Ojg4ODggbnZjci5pby9udmlkaWEvcHl0b3JjaDoyMy4xMC1weTMgL3Vzci9sb2NhbC9iaW4vanVweXRlciBsYWIgLS1hbGxvdy1yb290IC0taXA9KiAtLXBvcnQ9ODg4OCAtLW5vLWJyb3dzZXIgLS1Ob3RlYm9va0FwcC50b2tlbj0nJyAtLU5vdGVib29rQXBwLmFsbG93X29yaWdpbj0nKicgLS1ub3RlYm9vay1kaXI9L3dvcmtzcGFjZQ==</pre> che corrisponde allo script seguente in formato testo normale: <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> ■ Immettere le proprietà di installazione del driver guest vGPU, ad esempio `vgpu-license` e `nvidia-portal-api-key`. ■ Specificare i valori per le proprietà necessarie per un ambiente disconnesso in base alle esigenze. Vedere Proprietà OVF delle macchine virtuali di deep learning.
Output	■ Registri di installazione per il driver guest della vGPU in `/var/log/vgpu-install.log`. Per verificare che il driver guest della vGPU sia installato, eseguire il comando `nvidia-smi`. ■ Registri dello script cloud-init in `/var/log/dl.log`. ■ Container PyTorch. Per verificare che il container PyTorch sia in esecuzione, eseguire i comandi `sudo docker ps -a` e `sudo docker logs container_id`. ■ Istanza di JupyterLab accessibile all'indirizzo `http://dl_vm_ip:8888` Nel terminale di JupyterLab, verificare che nel notebook siano disponibili le funzionalità seguenti: ■ Per verificare che JupyterLab possa accedere alla risorsa vGPU, eseguire `nvidia-smi`. ■ Per verificare che i pacchetti relativi a PyTorch siano installati, eseguire `pip show`.

TensorFlow

È possibile utilizzare una macchina virtuale di deep learning con una libreria TensorFlow per esplorare AI conversazionale, l'elaborazione del linguaggio naturale (NLP) e altri tipi di modelli AI in una macchina virtuale. Vedere la pagina di TensorFlow.

Dopo l'avvio, la macchina virtuale di deep learning avvia un'istanza di JupyterLab con i pacchetti TensorFlow installati e configurati.

Tabella 3-3. Immagine del container TensorFlow

Componente	Descrizione
Immagine del container	`nvcr.io/nvidia/tensorflow:ngc_image_tag` Ad esempio: `nvcr.io/nvidia/tensorflow:23.10-tf2-py3` Per informazioni sulle immagini del container TensorFlow supportate per le macchine virtuali di deep learning, vedere Note di rilascio di VMware Deep Learning VM.
Input necessari	Per distribuire un carico di lavoro TensorFlow, è necessario impostare le proprietà OVF per la macchina virtuale di deep learning nel modo seguente: ■ Utilizzare una delle proprietà seguenti specifiche per l'immagine di TensorFlow. ■ Script cloud-init. Codificarlo nel formato 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/
nvidia/tensorflow:ngc_image_tag /usr/local/bin/jupyter lab --allow-
```

Tabella 3-3. Immagine del container TensorFlow (continua)

Componente	Descrizione

```
root --ip=* --port=8888 --no-browser --NotebookApp.token='' --
NotebookApp.allow_origin='*' --notebook-dir=/workspace

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

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

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

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

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

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

Tabella 3-3. Immagine del container TensorFlow (continua)

Componente	Descrizione

```
      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"
    }
```

Ad esempio, per tensorflow:23.10-tf2-py3, specificare lo script seguente in formato base64:

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkCB3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCiAg
ICAKICAgIERFRkFVTFRfUkVHX1VSST0ibnZjci5pbyIKICAgIFJFR0lTVFJZX1VSSV9Q
QVRIPSQoZ3JlcCByZWdpc3RyeS11cmkgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2Vk
IC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIGlmIFtbIC16
ICIkUkVHSVNUUllfVVJJX1BBVEgiIF1dOyB0aGVuCiAgICAgICMgSWYgUkVHSVNUUllf
VVJJX1BBVEggaXMgbnVsbCBvciBlbXB0eSwgdXNlIHRoZSBkZWZhdWx0IHZhbHVlCiAg
ICAgIFJFR0lTVFJZX1VSSV9QQVRIPSRERUZBVUxUX1JFR19VUkkKICAgICAgZWNobyAi
UkVHSVNUUllfVVJJX1BBVEggd2FzIGVtcHR5LiBVc2luZyBkZWZhdWx0OiAkUkVHSVNU
UllfVVJJX1BBVEgiCiAgICBmaQogICAgICAjIElmIFJFR0lTVFJZX1VSSV9QQVRI
IGNvbnRhaW5zICcvJywgZXh0cmFjdCB0aGUgVVJJIHBhcnQKICAgIGlmIFtbICRSRUdJ
U1RSWV9VUklfUEFUSCA9PSAqIi8iKiBdXTsgdGhlbgogICAgICBSRUdJU1RSWV9VUk9
JChlY2hvICIkUkVHSVNUUllfVVJJX1BBVEgiIHwgY3V0IC1kJy8nIC1mMSkKICAgIGVs
c2UKICAgICAgUkVHSVNUUllfVVJJX1BBVEg9JPSRSRUdJU1RSWV9VUklfUEFUSAogICAgZmkKICAK
ICAgIFJFR0lTVFJZX1VRVJOQU1FPSQoZ3JlcCByZWdpc3RyeS11c2VyIC9vcHQvZGx2
bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9c
MS9wJykKICAgIFJFR0lTVFJZX1BBU1NXT1JEPSQoZ3JlcCByZWdpc3RyeS1wYXNzd2Qg
L29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14i
XSpcKS4qL1wxL3AnKQogICAgaWYgW1sgLW4gIiRSRUdJU1RSWV9VU0VSTkFNRSIgJiYg
LW4gIiRSRUdJU1RSWV9QQVNTV09SRCIgXV07IHRoZW4KICAgICAgZG9ja2VyIGxvZ2lu
IC11ICRSRUdJU1RSWV9VU0VSTkFNRSAtcCAkUkVHSVNUUllfUEFTU1dPUkQgJFJFR0lT
VFJZX1VSSQogICAgZWxzZQogICAgICBlY2hvICJXYXJuaW5nOiB0aGUgcmVnaXN0cnkn
cyB1c2VybmFtZSBhbmQgcGFzc3dvcmQgYXJlIGludmFsaWQsIFNraXBwaW5nIERvY2tl
ciBsb2dpbi4iCiAgICBmaQogICAgICAgICBkb2NrZXIgcnVuIC1kIC0tZ3B1cyBhbGwg
LXAgODg4ODo4ODg4ICRSRUdJU1RSWV9VUklfUEFUSC9udmlkaWEvdGVuc29yZmxvdzoy
My4xMC10ZjItcHkzIC91c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9v
dCAtLWlwPSogLS1wb3J0PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAudG9r
ZW49JycgLS1Ob3RlYm9va0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGly
```

Tabella 3-3. Immagine del container TensorFlow (continua)

Componente	Descrizione


```
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
```


che corrisponde allo script seguente in formato testo normale:

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

Tabella 3-3. Immagine del container TensorFlow (continua)

Componente	Descrizione
	#!/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 (NF > 1) print $1; else print ""}')

Tabella 3-3. Immagine del container TensorFlow (continua)

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

Tabella 3-3. Immagine del container TensorFlow (continua)

Componente	Descrizione
	<pre> echo "Info: docker and system environment are now configured to use the proxy settings" }</pre> ■ Immagine one-liner. Codificarlo nel formato 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> Ad esempio, per tensorflow:23.10-tf2-py3, specificare lo script seguente in formato base64: <pre>ZG9ja2VyIHJ1biAtZCAtcCA4ODg4Ojg4ODggbnZjci5pby9udmlkaWEvdGVuc29yZmxvdzoyMy4xMC10ZjItcHkzIC91c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9vdCAtLWlwPSogLS1wb3J0PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAudG9rZW49JycgLS1Ob3RlYm9va0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGlyPS93b3Jrc3BhY2U=</pre> che corrisponde allo script seguente in formato testo normale: <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> ■ Immettere le proprietà di installazione del driver guest vGPU, ad esempio `vgpu-license` e `nvidia-portal-api-key`. ■ Specificare i valori per le proprietà necessarie per un ambiente disconnesso in base alle esigenze. Vedere Proprietà OVF delle macchine virtuali di deep learning.
Output	■ Registri di installazione per il driver guest della vGPU in `/var/log/vgpu-install.log`. Per verificare che il driver guest della vGPU sia installato, accedere alla macchina virtuale tramite SSH ed eseguire il comando `nvidia-smi`. ■ Registri dello script cloud-init in `/var/log/dl.log`. ■ Container TensorFlow. Per verificare che il container TensorFlow sia in esecuzione, eseguire i comandi `sudo docker ps -a` e `sudo docker logs container_id`. ■ Istanza di JupyterLab a cui è possibile accedere in `http://dl_vm_ip:8888`. Nel terminale di JupyterLab, verificare che nel notebook siano disponibili le funzionalità seguenti: ■ Per verificare che JupyterLab possa accedere alla risorsa vGPU, eseguire `nvidia-smi`. ■ Per verificare che i pacchetti relativi a TensorFlow siano installati, eseguire `pip show`.

DCGM Exporter

È possibile utilizzare una macchina virtuale di deep learning con Data Center GPU Manager (DCGM) Exporter per monitorare l'integrità delle GPU e ottenerne le metriche utilizzate da un carico di lavoro DL, tramite NVIDIA DCGM, Prometheus e Grafana.

Vedere la pagina DCGM Exporter.

In una macchina virtuale di deep learning eseguire il container DCGM Exporter insieme a un carico di lavoro DL che esegue le operazioni di AI. Dopo l'avvio di Deep Learning VM, DCGM Exporter è pronto a raccogliere le metriche di vGPU ed esportare i dati in un'altra applicazione per ulteriore monitoraggio e visualizzazione. È possibile eseguire il carico di lavoro DL monitorato come parte del processo cloud-init o dalla riga di comando dopo l'avvio della macchina virtuale.

Tabella 3-4. Immagine del container DCGM Exporter

Componente	Descrizione
Immagine del container	`nvcr.io/nvidia/k8s/dcgm-exporter:`*`ngc_image_tag`* Ad esempio: `nvcr.io/nvidia/k8s/dcgm-exporter:3.2.5-3.1.8-ubuntu22.04` Per informazioni sulle immagini di container di DCGM Exporter supportate per le macchine virtuali di deep learning, vedere Note di rilascio di VMware Deep Learning VM.
Input necessari	Per distribuire un carico di lavoro DCGM Exporter, è necessario impostare le proprietà OVF per la macchina virtuale di deep learning nel modo seguente: ■ Utilizzare una delle seguenti proprietà specifiche dell'immagine di DCGM Exporter. ■ Script cloud-init. Codificarlo nel formato 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
$REGISTRY_URI_PATH/nvidia/k8s/dcgm-exporter:ngc_image_tag
```

Tabella 3-4. Immagine del container DCGM Exporter (continua)

Componente	Descrizione

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

Tabella 3-4. Immagine del container DCGM Exporter (continua)

Componente	Descrizione

```
      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"
    }
```

Ad esempio, per una macchina virtuale di deep learning in cui è preinstallata un'istanza di DCGM Exporter dcgm-exporter:3.2.5-3.1.8-ubuntu22.04, specificare lo script seguente in formato base64

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCiAg
ICAKICAgIERFRkFVTFRfUkVHX1VSST0ibnZjci5pbyIKICAgIFJFR0lTVFJZX1VSSV9Q
QVRIPSQoZ3JlcCByZWdpc3RyeS11cmkgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2Vk
IC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIGlmIFtbIC16
ICIkUkVHSVNUUllfVVJJX1BBVEgiIF1dOyB0aGVuCiAgICAgICMgSWYgUkVHSVNUUllf
VVJJX1BBVEggaXMgbnVsbCBvciBlbXB0eSwgdXNlIHRoZSBkZWZhdWx0IHZhbHVlCiAg
ICAgIFJFR0lTVFJZX1VSSV9QQVRIPSRERUZBVUxUX1JFR19VUkkKICAgICAgZWNobyAi
UkVHSVNUUllfVVJJX1BBVEggd2FzIGVtcHR5LiBVc2luZyBkZWZhdWx0OiAkUkVHSVNU
UllfVVJJX1BBVEgiCiAgICBmaQogICAgICAjIElmIFJFR0lTVFJZX1VSSV9QQVRI
IGNvbnRhaW5zICcvJywgZXh0cmFjdCB0aGUgVVJJIHBhcnQKICAgIGlmIFtbICRSRUdJ
U1RSWV9VUklfUEFUSCA9PSAqIi8iKiBdXTsgdGhlbgogICAgICBSRUdJU1RSWV9VUkk9
JChlY2hvICIkUkVHSVNUUllfVVJJX1BBVEgiIHwgY3V0IC1kJy8nIC1mMSkKICAgIGVs
c2UKICAgICAgUkVHSVNUUllfVVJJX1BBVEg9IiIKICAgIGZpCiAgICBSRUdJU1RSWV9VUklfUEFUSAogICAgZmkKICAK
ICAgIFJFR0lTVFJZX1VTRVJOQU1FPSQoZ3JlcCByZWdpc3RyeS11c2VyIC9vcHQvZGx2
bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9c
MS9wJykKICAgIFJFR0lTVFJZX1BBU1NXT1JEPSQoZ3JlcCByZWdpc3RyeS1wYXNzd2Qg
L29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14i
XSpcKS4qL1wxL3AnKQogICAgaWYgW1sgLW4gIiRSRUdJU1RSWV9VU0VSTkFNRSIgJiYg
LW4gIiRSRUdJU1RSWV9QQVNTV09SRCIgXV07IHRoZW4KICAgICAgZG9ja2VyIGxvZ2lu
IC11ICRSRUdJU1RSWV9VU0VSTkFNRSAtcCAkUkVHSVNUUllfUEFTU1dPUkQgJFJFR0lT
VFJZX1VSSQogICAgZWxzZQogICAgICBlY2hvICJXYXJuaW5nOiB0aGUgcmVnaXN0cnkn
cyB1c2VybmFtZSBhbmQgcGFzc3dvcmQgYXJlIGludmFsaWQsIFNraXBwaW5nIERvY2tl
ciBsb2dpbi4iCiAgICBmaQoKICAgIGRvY2tlciBydW4gLWQgLS1ncHVzIGFsbCAtLWNh
cC1hZGQgU1lTX0FETUlOIC0tcm0gLXAgOTQwMDo5NDAwICRSRUdJU1RSWV9VUklfUEFU
SC9udmlkaWEvazhzL2RjZ20tZXhwb3J0ZXI6My4yLjUtMy4xLjgtdWJ1bnR1MjIuMDQK
Ci0gcGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1Jwog
IGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAg
ICBlY2hvICJFcnJvcjogJDEiID4mMgogICAgICB2bXRvb2xzZCAtLWNtZCAiaW5mby1z
```

Tabella 3-4. Immagine del container DCGM Exporter (continua)

Componente	Descrizione
	ZXQgZ3Vlc3RpbmZvLnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIGZhbHNlLCBE TFdvcmtsb2FkRmFpbHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgfQoKICAgIGNoZWNr X3Byb3RvY29sKCkgewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQK ICAgICAgbG9jYWwgc3VwcG9ydGVkX3Byb3RvY29scz0oIiRAIikKICAgICAgaWYgW1sg LW4gIiR7cHJveHlfdXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm90b2NvbD0k KGVjaG8gIiR7cHJveHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBw cmludCAkMTsgZWxzZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogcm90b2Nv bCIgXTsgdGhlbgogICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMgcHJvdG9jb2wgcHJv dmlkZWQuIFNraXBwaW5nIHByb3RvY29sIGNoZWNrLiKICAgICAgICAgIHJldHVybiAw CiAgICAgICAgZmkKICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQog ICAgICAgIGZvciB2YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwog ICAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9IiBdXTsgdGhlbgog ICAgICAgICAgICBwcm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgIGJyZWFr CiAgICAgICAgICBmaQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2Nv bF9pbmNsdWRlZH0iID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0 ICJVbnN1cHBvcnRlZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1cHBvcnRlZCBwcm90 b2NvbHMgYXJlOiAke3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgIGZpCiAg ICAgIGZpCiAgICB9CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xz CiAgICBzZXRfcHJveHkoKSB7CiAgICAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9 KCIkQCIpCgogICAgICBDT05GSUdfSlNPTl9CQVNFNjQ9JChncmVwICdjb25maWctanNv bicgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9Ilwo W14iXSpcKS4qL1wxL3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdf SlNPTl9CQVNFNjR9IHwgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgSFRUUF9QUk9YWV9V Ukw9JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0dHBfcHJveHkgLy8g ZW1wdHknKQogICAgICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09O fSIgfCBqcSAtciAnLmh0dHBzX3Byb3h5IC8vIGVtcHR5KyJkKICAgICAgaWYgW1sgJD8g LW5lIDAgfHwgKC16ICIke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJP WFlfVVJMfSIpIF1dOyB0aGVuCiAgICAgICAgZWNobyAiSW5mbzogVGhlIGNvbmZpZy1q c29uIHdhcyBwYXJzZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kLiIK ICAgICAgICByZXR1cm4gMAogICAgICBmaQoKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7 SFRUUF9QUk9YWV9VUkx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAg Y2hlY2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVkX3By b3RvY29sc1tAXX0iCgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldGMv ZW52aXJvbm1lbnQ7IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0k e0hUVFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBT X1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJPWFlf VVJMfQogICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH0K ICAgICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0 Yy9lbnZpcm9ubWVudAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAgICAg IGZpCiAgICAgIAogICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJveHkK ICAgICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5k CiAgICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJP WFk9JHtIVFRQX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BS T1hZPSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25tZW50PVwiTk9fUFJP WFk9bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9j a2VyLnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVs b2FkCiAgICAgICAgIHN5c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAiSW5m bzogZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1cmVk IHRvIHVzZSB0aGUgcHJveHkgc2V0dGluZ3MiCiAgICB9

che corrisponde allo script seguente in formato testo normale:

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

Tabella 3-4. Immagine del container DCGM Exporter (continua)

Componente	Descrizione

```
    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."
          return 0
```

Tabella 3-4. Immagine del container DCGM Exporter (continua)

Componente	Descrizione
	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

Tabella 3-4. Immagine del container DCGM Exporter (continua)

Componente	Descrizione
	echo "Info: docker and system environment are now configured to use the proxy settings" } **Nota** Nello script cloud-init è inoltre possibile aggiungere le istruzioni per l'esecuzione del carico di lavoro DL di cui si desidera misurare le prestazioni della GPU con DCGM Exporter. ■ Immagine one-liner. Codificarlo nel formato 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` Ad esempio, per dcgm-exporter:3.2.5-3.1.8-ubuntu22.04, specificare lo script seguente in formato base64: `ZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tY2FwLWFkZCBTWVNfQURNSU4gLS1ybSAtcCA5NDAwOjk0MDAgbnZjci5pby9udmlkaWEvazhzL2RjZ20tZXhwb3J0ZXI6My4yLjUtMy4xLjgtdWJ1bnR1MjIuMDQ=` che corrisponde allo script seguente in formato testo normale: `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` ■ Immettere le proprietà di installazione del driver guest vGPU, ad esempio `vgpu-license` e `nvidia-portal-api-key`. ■ Specificare i valori per le proprietà necessarie per un ambiente disconnesso in base alle esigenze. Vedere Proprietà OVF delle macchine virtuali di deep learning.
Output	■ Registri di installazione per il driver guest della vGPU in `/var/log/vgpu-install.log`. Per verificare che il driver guest della vGPU sia installato, accedere alla macchina virtuale tramite SSH ed eseguire il comando `nvidia-smi`. ■ Registri dello script cloud-init in `/var/log/dl.log`. ■ DCGM Exporter a cui è possibile accedere all'indirizzo `http://`*`dl_vm_ip`*`:9400`. Nella macchina virtuale di deep learning, eseguire quindi un carico di lavoro DL e visualizzare i dati in un'altra macchina virtuale utilizzando Prometheus all'indirizzo `http://`*`visualization_vm_ip`*`:9090` e Grafana all'indirizzo `http://`*`visualization_vm_ip`*`:3000`.

Esecuzione di un carico di lavoro DL nella macchina virtuale di deep leaning

Eseguire il carico di lavoro DL per cui si desidera raccogliere le metriche vGPU ed esportare i dati in un'altra applicazione per ulteriori informazioni di monitoraggio e visualizzazione.

1 Accedere alla macchina virtuale di deep learning come **vmware** tramite SSH.

2 Aggiungere l'account utente **vmware** al gruppo **docker** eseguendo il comando seguente.

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

3 Eseguire il container per il carico di lavoro DL, estraendolo dal catalogo NVIDIA NGC o da un registro di container locale.

   Ad esempio, per eseguire il comando seguente per l'esecuzione dell'immagine tensorflow:23.10-tf2-py3 da 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 Iniziare a utilizzare il carico di lavoro DL per lo sviluppo di AI.

Installazione di Prometheus e Grafana

È possibile visualizzare e monitorare le metriche della vGPU dalla macchina virtuale di DCGM Exporter in una macchina virtuale che esegue Prometheus e Grafana.

1 Creare una macchina virtuale di visualizzazione in cui è installato Docker Community Engine.

2 Connettersi alla macchina virtuale tramite SSH e creare un file YAML per 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 Creare un percorso dati.

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

4 Creare un file di composizione Docker per installare Prometheus e 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 Avviare i container di Prometheus e Grafana.

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

Visualizzazione delle metriche della vGPU in Prometheus

È possibile accedere a Prometheus all'indirizzo `http://visualization-vm-ip:9090`. È possibile visualizzare le seguenti informazioni sulla vGPU nell'interfaccia utente di Prometheus:

Informazioni	Sezione dell'interfaccia utente
Metriche della vGPU non elaborate dalla macchina virtuale di deep learning	**Stato > Destinazione** Per visualizzare le metriche della vGPU non elaborate dalla macchina virtuale di deep learning, fare clic sulla voce dell'endpoint.
Espressioni del grafico	1 Nella barra di navigazione principale, fare clic sulla scheda **Grafico**. 2 Immettere un'espressione e fare clic su **Esegui**

Per ulteriori informazioni sull'utilizzo di Prometheus, vedere la documentazione di Prometheus.

Visualizzazione delle metriche in Grafana

Impostare Prometheus come origine dati per Grafana e visualizzare le metriche della vGPU dalla macchina virtuale di deep learning in un dashboard.

1 Accedere a Grafana all'indirizzo `http://visualization-vm-ip:3000` utilizzando il nome utente predefinito **admin** e la password `admin`.

2 Aggiungere Prometheus come prima origine dati connettendosi a *visualization-vm-ip* nella porta 9090.

3 Creare un dashboard con le metriche della vGPU.

Per ulteriori informazioni sulla configurazione di un dashboard utilizzando un'origine dati Prometheus, vedere la documentazione di Grafana.

Triton Inference Server

È possibile utilizzare una macchina virtuale di deep learning con Triton Inference Server per caricare un repository di modelli e ricevere richieste di inferenza.

Vedere la pagina Triton Inference Server.

Tabella 3-5. Immagine del container Triton Inference Server

Componente	Descrizione
Immagine del container	`nvcr.io/nvidia/tritonserver:ngc_image_tag` Ad esempio: `nvcr.io/nvidia/tritonserver:23.10-py3` Per informazioni sulle immagini dei container Triton Inference Server supportate per le macchine virtuali di deep learning, vedere Note di rilascio di VMware Deep Learning VM.
Input necessari	Per distribuire un carico di lavoro Triton Inference Server, è necessario impostare le proprietà OVF per la macchina virtuale di deep learning nel modo seguente: ■ Utilizzare una delle proprietà seguenti specifiche per l'immagine di Triton Inference Server. ■ Script cloud-init. Codificarlo nel formato 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 --rm -p 8000:8000 -p
8001:8001 -p 8002:8002 -v /home/vmware/model_repository:/models
```

Tabella 3-5. Immagine del container Triton Inference Server (continua)

Componente	Descrizione
	`$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 "${HTTPS_PROXY_URL}") ]]; then` `  echo "Info: The config-json was parsed, but no proxy settings were found."` `  return 0` `  fi` `  check_protocol "${HTTP_PROXY_URL}" "${supported_protocols[@]}"`

Tabella 3-5. Immagine del container Triton Inference Server (continua)

Componente	Descrizione

```
      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"
    }
```

Ad esempio, per tritonserver:23.10-py3, specificare lo script seguente in formato base64:

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkb3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCgog
ICAgREVGQVVMVF9SRUdfVVJJPSJudmNyLmlvIgogICAgUkVHSVNUUllfVVJJX1BBVEg9
JChncmVwIHJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4g
J3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCgogICAgaWYgWlsgLXogIiRS
RUdJU1RSWV9VUklfUEFUSCIgXV07IHRoZW4KICAgICAgIyBJZiBSRUdJU1RSWV9VUklf
UEFUSCBpcyBudWxsIG9yIGVtcHR5LCB1c2UgdGhlIGRlZmF1bHQgdmFsdWUKICAgICAg
UkVHSVNUUllfVVJJX1BBVEg9JERFRkFVTFRfUkVHX1VSSQogICAgICBlY2hvICJSRUdJ
U1RSWV9VUklfUEFUSCB3YXMgZW1wdHkuIFVzaW5nIGRlZmF1bHQ6ICRSRUdJU1RSWV9V
UklfUEFUSCIKICAgIGZpCiAgICAKICAgICMgSWYgUkVHSVNUUllfVVJJX1BBVEggY29u
dGFpbnMgJy8nLCBleHRyYWN0IHRoZSBVUkkgcGFydAogICAgaWYgWlsgJFJFR0lTVFJZ
X1VSSV9QQVRIID09ICoiLyIqIF1dOyB0aGVuCiAgICAgIFJFR0lTVFJZX1VSST0kKGVj
aG8gIiRSRUdJU1RSWV9VUklfUEFUSCIgfCBjdXQgLWQnLycgLWYxKQogICAgZWxzZQog
ICAgICBSRUdJU1RSWV9VUkk9JFJFR0lTVFJZX1VSSV9QQVRICiAgICBmaQogICAgICAg
UkVHSVNUUllfVVNFUk5BTUU9JChncmVwIHJlZ2lzdHJ5LXVzZXIgL29wdC9kbHZtL292
Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3An
KQogICAgUkVHSVNUUllfUEFTU1dPUkQ9JChncmVwIHJlZ2lzdHJ5LXBhc3N3ZCAvb3B0
L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwp
LiovXDEvcCcpCiAgICBpZiBbWyAtbiAiJFJFR0lTVFJZX1VTRVJOQU1FIiAmJiAtbiAi
JFJFR0lTVFJZX1BBU1NXT1JEIiBdXTsgdGhlbgogICAgICBkb2NrZXIgbG9naW4gLXUg
JFJFR0lTVFJZX1VTRVJOQU1FIC1wICRSRUdJU1RSWV9QQVNTV09SRCAkUkVHSVNUUllf
VVJJCiAgICBlbHNlCiAgICAgIGVjaG8gIldhcm5pbmc6IHRoZSByZWdpc3RyeSdzIHVz
ZXJuYW1lIGFuZCBwYXNzd29yZCBhcmUgaW52YWxpZCwgU2tpHBpbmcgRG9ja2VyIGxv
Z2luLiIKICAgIGZpCgogICAgZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tcm0gLXAg
ODAwMDo4MDAwIC1wIDgwMDE6ODAwMSAtcCA4MDAyOjgwMDIgLXYgL2hvbWUvdm13YXJl
L21vZGVsX3JlcG9zaXRvcnk6L21vZGVscyAkUkVHSVNUUllfVVJJX1BBVEgvbnZpZGlh
L3RyaXRvbnNlcnZlcjoyMy4xMC1weTMgdHJpdG9uc2VydmVyIC0tbW9kZWwtcmVwb3Np
dG9yeT0vbW9kZWxzIC0tbW9kZWwtY29udHJvbC1tb2RlPXBvbGwKCi0gcGF0aDogL29w
dC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwK
```

Tabella 3-5. Immagine del container Triton Inference Server (continua)

Componente	Descrizione
	ICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAgICBlY2hvICJFcnJv cjogJDEiID4mMgogICAgICB2bXRvb2xzZCAtLWNtZCAiaW5mby1zZXQgZ3Vlc3RpbmZv LnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIGZhbHNlLCBETFdvcmtsb2FkRmFp bHVyZSwgJDEiCiAgICAgICAgV4aXQgMQogICAgfQoKICAgIGNoZWNrX3Byb3RvY29sKCkg ewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQKICAgICAgbG9jYWwg c3VwcG9ydGVkX3Byb3RvY29scz0oIiRAIikKICAgICAgaWYgW1sgLW4gIiR7cHJveHlf dXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm90b2NvbD0kKGVjaG8gIiR7cHJv eHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBwcmludCAkMTsgZWxz ZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2NvbCIgXTsgdGhlbgog ICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMgcHJvdG9jb2wgcHJvdmlkZWQuIFNraXBw aW5nIHByb3RvY29sIGNoZWNrLiKICAgICAgICAgIHJldHVybiAwCiAgICAgICAgZmkK ICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQogICAgICAgIGZvciB2 YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwogICAgICAgICAgaWYg W1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9IiBdXTsgdGhlbgogICAgICAgICAgICBw cm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgIGJyZWFrCiAgICAgICAgICBm aQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2NvbF9pbmNsdWRlZH0i ID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0ICJVbnN1cHBvcnRl ZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1cHBvcnRlZCBwcm90b2NvbHMgYXJlOiAk e3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgIGZpCiAgICAgICAgZpCiAgICB9 CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xzCiAgICBzZXRfcHJv eHkoKSB7CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIpCgogICAg ICBDT05GSUdfSlNPTl9CQVNFNjQ9JChncmVwICdjb25maWctanNvbicgL29wdC9kbHZt L292ZillbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wx L3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdfSlNPTl9CQVNFNjR9 IHwgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgSFRUUF9QUk9YWV9VUkw9JChlY2hvICIk e0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0dHBfcHJveHkgLy8gZW1wdHknKQogICAg ICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqcSAtciAn Lmh0dHBzX3Byb3h5IC8vIGVtcHR5JykKICAgICAgaWYgW1sgJD8gLW5lIDAgfHwgKC16 ICIke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJPWFlfVVJMfSIpIF1d OyB0aGVuCiAgICAgICAgZWNobyAiSW5mbzogVGhlIGNvbmZpZy1qc29uIHdhcyBwYXJz ZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kLiIKICAgICAgICByZXR1 cm4gMAogICAgICBmaQoKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUF9QUk9YWV9V Ukx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAgY2hlY2tfcHJvdG9j b2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0i CgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldGMvZW52aXJvbm1lbnQ7 IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0ke0hUVFBfUFJPWFlf VVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBTX1BST1hZX1VSTH0K ICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJPWFlfVVJMfQogICAgICAg IGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH0KICAgICAgICBleHBv cnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0Yy9lbnZpcm9ubWVu dAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAgICAgICAgZpCiAgICAgICAg ICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJveHkKICAgICAgICAgbWtkaXIg LXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5kCiAgICAgICAgICAg IltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJPWFk9JHtIVFRQX1BS T1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BST1hZPSR7SFRUUFNf UFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25tZW50PVwiTk9fUFJPWFk9bG9jYWxob3N0 LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9ja2VyLnNlcnZpY2Uu ZC9wcm94eS5jb25mCiAgICAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVsb2FkCiAgICAg c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAiSW5mbzogZG9ja2VyIGFu ZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1cmVkIHRvIHVzZSB0aGUg cHJveHkgc2V0dGluZ3MiCiAgICB9

che corrisponde allo script seguente in formato testo normale:

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

Tabella 3-5. Immagine del container Triton Inference Server (continua)

Componente	Descrizione

```
    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
(NF > 1) print $1; else print ""}')
        if [ -z "$protocol" ]; then
```

Tabella 3-5. Immagine del container Triton Inference Server (continua)

Componente	Descrizione

```
          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
```

Tabella 3-5. Immagine del container Triton Inference Server (continua)

Componente	Descrizione
	echo "Info: docker and system environment are now configured to use the proxy settings" } ■ Immagine one-liner codificata nel formato base64 `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` Ad esempio, per tritonserver:23.10-py3, specificare lo script seguente in formato base64: `ZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tcm0gLXA4MDAwOjgwMDAgLXA4MDAxOjgwMDEgLXA4MDAyOjgwMDIgLXYgL2hvbWUvdm13YXJlL21vZGVsX3JlcG9zaXRvcnk6L21vZGVscyBudmNyLmlvL252aWRpYS90cml0b25zZXJ2ZXI6MjMuMTAtcHkzIHRyaXRvbnNlcnZlciAtLW1vZGVsLXJlcG9zaXRvcnk9L21vZGVscyAtLW1vZGVsLWNvbnRyb2wtbW9kZT1wb2xs` che corrisponde allo script seguente in formato testo normale: `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` ■ Immettere le proprietà di installazione del driver guest vGPU, ad esempio `vgpu-license` e `nvidia-portal-api-key`. ■ Specificare i valori per le proprietà necessarie per un ambiente disconnesso in base alle esigenze. Vedere Proprietà OVF delle macchine virtuali di deep learning.
Output	■ Registri di installazione per il driver guest della vGPU in `/var/log/vgpu-install.log`. Per verificare che il driver guest della vGPU sia installato, accedere alla macchina virtuale tramite SSH ed eseguire il comando `nvidia-smi`. ■ Registri dello script cloud-init in `/var/log/dl.log`. ■ Container Triton Inference Server. Per verificare che il container Triton Inference Server sia in esecuzione, eseguire i comandi `sudo docker ps -a` e `sudo docker logs container_id`. Il repository di modelli per Triton Inference Server è in `/home/vmware/model_repository`. Inizialmente, il repository di modelli è vuoto e il registro iniziale dell'istanza di Triton Inference Server indica che non è stato caricato alcun modello.

Creazione di un repository di modelli

Per caricare il modello per l'inferenza del modello, eseguire i passaggi seguenti:

1 Creare il repository di modelli per il proprio modello.

Vedere la documentazione relativa al repository di modelli di NVIDIA Triton Inference Server.

2 Copiare il repository di modelli in `/home/vmware/model_repository` in modo che Triton Inference Server possa caricarlo.

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

Invio di richieste di inferenza del modello

1 Verificare che Triton Inference Server sia integro e che i modelli siano pronti eseguendo questo comando nella console della macchina virtuale di deep learning.

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

2 Inviare una richiesta al modello eseguendo questo comando nella macchina virtuale di deep learning.

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

Per ulteriori informazioni sull'utilizzo di Triton Inference Server, vedere la documentazione relativa al repository di modelli di NVIDIA Triton Inference Server.

NVIDIA RAG

È possibile utilizzare una macchina virtuale di deep learning per creare soluzioni RAG (Retrieval Augmented Generation) con un modello Llama2.

Vedere la documentazione NVIDIA RAG Applications Docker Compose (richiede autorizzazioni dell'account specifiche).

Tabella 3-6. Immagine del container NVIDIA RAG

Componente	Descrizione
Immagini e modelli di container	`rag-app-text-chatbot.yaml` nella pipeline di NVIDIA RAG di esempio. Per informazioni sulle applicazioni container NVIDIA RAG supportate per le macchine virtuali di deep learning, vedere Note di rilascio di VMware Deep Learning VM.
Input necessari	Per distribuire un carico di lavoro NVIDIA RAG, è necessario impostare le proprietà OVF per la macchina virtuale di deep learning nel modo seguente: ■ Immettere uno script cloud-init. Codificarlo nel formato base64. Ad esempio, per la versione 24.03 di NVIDIA RAG, specificare lo script seguente: I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBwLnN oCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCi AgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0cmFwICdlc nJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3JrbG9hZCInIEVS UgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiCgogICAgY2F0IDw8RU9GID4gL29wdC9 kbHZtL2NvbmZpZy5qc29uCiAgICB7CiAgICAgICJfY29tbWVudCI6ICJUaGlzIHByb3ZpZG VzIGRlZmF1bHQgc3VwcG9ydCBmb3IgUkFHOiBUZW5zb3JSVCBpbmZlcmVuY2UsIGxsYW1hM i0xM2IgbW9kZWwsIGFuZCBIMTAweDIgR1BVIiwKICAgICAgInJhZyI6IHsKICAgICAgICAi b3JnX25hbWUiOiAiY29jZndnYThqcTJjIiwKICAgICAgICAib3JnX3RlYW1fbmFtZSI6ICJ uby10ZWFtIiwKICAgICAgICAicmFnX3JlcG9fbmFtZSI6ICJudmlkaWEvcGFpZiIsCiAgIC AgICAgImxsbV9yZXBvX25hbWUiOiAibnZpZGlhL25pbSIsCiAgICAgICAgImVtYmVkX3Jlc G9fbmFtZSI6ICJudmlkaWEvbmVtby1yZXRyaWV2ZXIiLAogICAgICAgICJyYWdfbmFtZSI6 ICJyYWctZG9ja2VyLWNvbXBvc2UiLAogICAgICAgICJyYWdfdmVyc2lvbiI6ICIyNC4wMyI sCiAgICAgICAgImVtYmVkX25hbWUiOiAibnYtZW1iZWQtcWEiLAogICAgICAgICJlbWJlZF 90eXBlIjogIk5WLUVtYmVkLVFBIiwKICAgICAgICAiZW1iZWRfdmVyc2lvbiI6ICI0IiwKI CAgICAgICAiaW5mZXJlbmNlX3R5cGUiOiAidHJ0IiwKICAgICAgICAibGxtX25hbWUiOiAi bGxhbWEyLTEzYi1jaGF0IiwKICAgICAgICAibGxtX3ZlcnNpb24iOiAiaDEwMHgyX2ZwMTZ fMjQuMDIiLAogICAgICAgICJudW1fZ3BlIjogIjIiLAogICAgICAgICJoZl90b2tlbiI6IC JodWdnaW5nZmFjZSB0b2tlbiB0byBwdWxsIGxsbSBtb2RlbCwgdXBkYXRlIHdoZW4gdXNpb mcgdmxsbSBpbmZlcmVuY2UiLAogICAgICAgICJoZl9yZXBvIjogImh1Z2dpbmdmYWNlIGxs bSBtb2RlbCByZXBvc2l0b3J5LCB1cGRhdGUgd2hlbiB1c2luZyB2bGxtIGluZmVyZW5jZSI KICAgICAgfQogICAgfQogICAgRU9GCiAgICBDT05GSUdfSlNPTj0kKGNhdCAiL29wdC9kbH ZtL2NvbmZpZy5qc29uIikKICAgIElORkVSRU5DRV9UWVBFPSQoZWNobyAiJHtDT05GSUdfS lNPTn0iIHwganEgLXIgJy5yYWcuaW5mZXJlbmNlX3R5cGUnKQogICAgaWYgWyAiJHtJTkZF UkVOQ0VfVFlQRX0iID0gInRydCIgXTsgdGhlbgogICAgICByZXF1aXJlZF92YXJzPSgiT1J HX05BTUUiICJPUkdfVEVBTV9OQU1FIiAiUkFHX1JFUE9fTkFNRSIgIkxMTV9SRVBPX05BTU UiICJFTUJFRF9SRVBPX05BTUUiICJSQUdfTkFNRSIgIlJBR19WRVJTSU9OIiAiRU1CRURfT kFNRSIgIkVNQkVEX1RZUEUiICJFTUJFRF9WRVJTSU9OIiAiTExNX05BTUUiICJMTE1fVkVS U0lPTiIgIk5VTV9HUFUiKQogICAgZWxpZiBbICIke0lORkVSRU5DRV9UWVBfSIgPSAidmx sbSIgXTsgdGhlbgogICAgICByZXF1aXJlZF92YXJzPSgiT1JHX05BTUUiICJPUkdfVEVBTV 9OQU1FIiAiUkFHX1JFUE9fTkFNRSIgIkxMTV9SRVBPX05BTUUiICJFTUJFRF9SRVBPX05BT UUiICJSQUdfTkFNRSIgIlJBR19WRVJTSU9OIiAiRU1CRURfTkFNRSIgIkVNQkVEX1RZUEUi ICJFTUJFRF9WRVJTSU9OIiAiTExNX05BTUUiICJOVU1fR1BVIiAiSEZfVE9LRU4iICJIRl9 SRVBPIikKICAgIGVsc2UKICAgICAgZXJyb3JfZXhpdCAiSW5mZXJlbmNlIHR5cGUgJyR7SU 5GRVJFTkNFX1RZUEV9JyBpcyBub3QgcmVjb2duaXplZC4gTm8gYWN0aW9uIHdpbGwgYmUgd GFrZW4uIgogICAgZmkKICAgIGZvciBpbmRleCBpbiAiJHshcmVxdWlyZWRfdmFyc1tAXX0i OyBkbwogICAgICBrZXk9IiR7cmVxdWlyZWRfdmFyc1skaW5kZXhdfSIKICAgICAganFfcXV lcnk9Ii5yYWcuJHtrZXksLH0gfCBzZWxlY3QgKC4hPW51bGwpIgogICAgICB2YWx1ZT0kG VjaG8gIiR7Q09ORklHX0pTT059IiB8IGpxIC1yICIke2pxX3F1ZXJ5fSIpCiAgICAgIGlmI FtbIC16ICIke3ZhbHVlfSIgXV07IHRoZW4gCiAgICAgICAgZXJyb3JfZXhpdCAiJHtrZXl9 IGlzIHJlcXVpcmVkIGJ1dCBub3Qgc2V0LiIKICAgICAgZWxzZQogICAgICAgIGV2YWwgJHt rZXl9PVwiIiR7dmFsdWV9IlwiCiAgICAgICAgIGZpCiAgICBkb25lCgogICAgUkFHX1VSST0iJH tSQUdfUkVQT19OQU1FfS8ke1JBR19OQU1FfToke1JBR19WRVJTSU9OfSIKICAgIEVNQkVEX 01PREVMX1VSST0iJHtFTUJFRF9SRVBPX05BTUV9LyR7RU1CRURfTkFNRX06JHtFTUJFRF9W RVJTSU9OfSIKCiAgICBOR0NfQ0xJX1ZFUlNJT049IjMuNDEuMiIKICAgIE5HQ19DTElfVVJ MPSJodHRwczovL2FwaS5uZ2MubnZpZGlhLmNvbS92Mi9yZXNvdXJjZXMvbnZpZGlhL25nYy 1hcHBzL25nY19jbGkvdmVyc2lvbnMvJHtOR0NfQ0xJX1ZFUlNJT059L2ZpbGVzL25nY2Nsa

Tabella 3-6. Immagine del container NVIDIA RAG (continua)

Componente	Descrizione


```
V9saW51eC56aXAiCgogICAgbWtkaXIgLXAgL29wdC9kYXRhCiAgICBjZCAvb3B0L2RhdGEK
CiAgICBpZiBbICEgLWYgLmZpbGVfZG93bmxvYWRlZCBdOyB0aGVuCiAgICAgICMgY2xlYW4
gdXAKICAgICAgcm0gLXJmIGNvbXBvc2UuZW52ICR7UkFHX05BTUV9KiAke0xMTV9OQU1FfS
ogbmdjKiAke0VNQkVEX05BTUV9KiAqLmpzb24gLmZpbGVfZG93bmxvYWRlZAoKICAgICAgI
yBpbnN0YWxsIG5nYy1jbGkKICAgICAgd2dldCAtLWNvbnRlbnQtZGlzcG9zaXRpb24gJHtO
R0NfQ0xJX1VSTH0gLU8gbmdjY2xpX2xpbnV4LnppcCAmJiB1bnppcCBuZ2NjbGlfbGludXg
uemlwCiAgICAgIGV4cG9ydCBQQVRIPWBwd2RgL25nYy1jbGk6JHtQQVRIfQoKICAgICAgQV
BJS0VZPSIiCiAgICAgIFJFR19VUkk9Im52Y3IuaW8iCgogICAgICBpZiBbWyAiJChncmVwI
HJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2
YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpIiA9PSAqIiR7UkVHX1VSSX0iKiBdXTsgdGhlbgo
gICAgICAgIEFQSUtFWT0kKGdyZXAgcmVnaXN0cnktcGFzc3dkIC9vcHQvZGx2bS9vdmYtZW
52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKICAgI
CAgZmkKCiAgICAgIGlmIFsgLXogIiR7QVBJS0VZfSIgXTsgdGhlbgogICAgICAgICAgZXJy
b3JfZXhpdCAiTm8gQVBJS0VZIGZvdW5kIgogICAgICBmaQoKICAgICAgIyBjb25maWcgbmd
jLWNsaQogICAgICBta2RpciAtcCB+Ly5uZ2MKCiAgICAgIGNhdCA8PCBFT0YgPiB+Ly5uZ2
MvY29uZmlnCiAgICAgIFtDVVJSRU5UXQogICAgICBhcGlrZXkgPSAke0FQSUtFWX0KICAgI
CAgZm9ybWF0X3R5cGUgPSBhc2NpaQogICAgICBvcmcgPSAke09SR19OQU1FfQogICAgICB0
ZWFtID0gJHtPUkdfVEVBTV9OQU1FfQogICAgICBhY2UgPSBuby1hY2UKICAgICAgIEVPRgoKICA
gICAgIyBuZ2MgZG9ja2VyIGxvZ2luCiAgICAgIGRvY2tlciBsb2dpbiBudmNyLmlvIC11IF
wkb2F1dGh0b2tlbiAtcCAke0FQSUtFWX0KCiAgICAgICMgZG9ja2VyaHViIGxvZ2luIGZvc
iBnZW5lcmFsIGNvbXBvbmVudHMsIGUuZy4gbWluaW8KICAgICAgRE9DS0VSSFVCX1VSST0k
KGdyZXAgcmVnaXN0cnktMi11cmkgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICd
zLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQogICAgICBET0NLRVJIVUJfVVNFUk
5BTUU9JChncmVwIHJlZ2lzdHJ5LTItdXNlciAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZ
WQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCiAgICAgIERPQ0tFUkhV
Ql9QQVNTV09SRD0kKGdyZXAgcmVnaXN0cnktMi1wYXNzd2QgL29wdC9kbHZtL292Zi1lbnY
ueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIC
AgaWYgW1sgLW4gIiR7RE9DS0VSSFVCX1VTRVJOQU1FfSIgJiYgLW4gIiR7RE9DS0VSSFVCX
1BBU1NXT1JEfSIgXV07IHRoZW4KICAgICAgICBkb2NrZXIgbG9naW4gLXUgJHtET0NLRVJI
VUJfVVNFUk5BTUV9IC1wICR7RE9DS0VSSFVCX1BBU1NXT1JEfQogICAgICBlbHNlCiAgICA
gICAgZWNobyAiV2FybmluZzogRG9ja2VySHViIG5vdCBsb2dpbiIKICAgICAgZmkKCiAgIC
AgICMgZ2V0IFJBRyBmaWxlcwogICAgICBuZ2MgcmVnaXN0cnkgcmVzb3VyY2UgZG93bmxvY
WQtdmVyc2lvbiAke1JBR19VUkl9CgogICAgICAjIGdldCBsbG0gbW9kZWwKICAgICAgaWYg
WyAiJHtJTkZFUkVOQ0VfVFlQRX0iID0gInRydCIgXTsgdGhlbgogICAgICAgICAgIExMTV9NT0R
FTF9VUkk9IiR7TExNX1JFUE9fTkFNRX0vJHtMTE1fTkFNRX06JHtMTE1fVkVSU0lPTn0iCi
AgICAgICAgbmdjIHJlZ2lzdHJ5IG1vZGVsIGRvd25sb2FkLXZlcnNpb24gJHtMTE1fTU9ER
UxfVVJJfQogICAgICAgIGNobW9kIC1SIG8rclggJHtMTE1fTkFNRX1fdiR7TExNX1ZFUlNJ
T059CiAgICAgICAgTExNX01PREVMX0ZPTERFUj0iL29wdC9kYXRhLyR7TExNX05BTUV9X3Y
ke0xMTV9WRVJTSU9OfSIKICAgICAgZWxpZiBbICIke0lORkVSRU5DRV9UWVBFfSIgPSAidm
xsbSIgXTsgdGhlbgogICAgICAgIHBpcCBpbnN0YWxsIGh1Z2dpbmdmYWNlX2h1YgogICAgI
CAgIGh1Z2dpbmdmYWNlLWNsaSBsb2dpbiAtLXRva2VuICR7SEZfVE9LRU59CiAgICAgICAg
aHVnZ2luZ2ZhY2UtY2xpIGRvd25sb2FkIC0tcmVzdW1lLWRvd25sb2FkICR7SEZfUkVQT30
vJHtMTE1fTkFNRX0gLS1sb2NhbC1kaXIgJHtMTE1fTkFNRX0gLS1sb2NhbC1kaXItdXNlLX
N5bWxpbmtzIEZhbHNlCiAgICAgICAgTExNX01PREVMX0ZPTERFUj0iL29wdC9kYXRhLyR7T
ExNX05BTUV9IgogICAgICAgIGNhdCA8PCBFT0YgPiAke0xMTV9NT0RFTF9GT0xERVJ9L21v
ZGVsX2NvbmZpZy55YW1sIAogICAgICAgIGVuZ2luZToKICAgICAgICAgIG1vZGVsOiAvbW9
kZWwtc3RvcmUKICAgICAgICAgIGVuZm9yY2VfZWFnZXI6IGZhbHNlCiAgICAgICAgICBtYX
hfY29udGV4dF9sZW5fdG9fY2FwdHVyZTogODE5MgogICAgICAgICAgbWF4X251bV9zZXFzO
iAyNTYKICAgICAgICAgIGR0eXBlOiBmbG9hdDE2CiAgICAgICAgICB0ZW5zb3JfcGFyYWxs
ZWxfc2l6ZTogJHtOVU1fR1BVfQogICAgICAgICAgZ3B1X21lbW9yeV91dGlsaXphdGlvbjo
gMC44CiAgICBFT0YKICAgICAgICBjaG1vZCAtUiBvK3JYICR7TExNX01PREVMX0ZPTERFUn
0KICAgICAgICBweXRob24zIC1jICJpbXBvcnQgeWFtbCwganNvbiwgc3lzOyBwcmludChqc
29uLmR1bXBzKHlhbWwuc2FmZV9sb2FkKHN5cy5zdGRpbi5yZWFkKCkpKSkiIDwgIiR7UkFH
X05BTUV9X3Yke1JBR19WRVJTSU9OfS9yYWctYXBwLXRleHQtY2hhdGJvdC55YW1sIj4gcmF
nLWFwcC10ZXh0LWNoYXRib3QuanNvbgogICAgICAgIGpxICcuc2VydmljZXMuIm5lbW9sbG
0taW5mZXJlbmNlIi5pbWFnZSA9ICJudmNyLmlvL252aWRpYS9uaW0vbmltX2xsbToyNC4wM
i1kYXkwIiB8CiAgICAgICAgICAgICAgIC5zZXJ2aWNlcy4ibmVtb2xsbS1pbmZlcmVuY2UiLmNv
bW1hbmQgPSAibmltX3ZsbG0gLS1tb2RlbF9uYW1lICR7TU9ERUxfTkFNRX0gLS1tb2RlbF9
jb25maWcgL21vZGVsLXN0b3JlL21vZGVsX2NvbmZpZy55YW1sIiB8CiAgICAgICAgICAgICAgIC
5zZXJ2aWNlcy4ibmVtb2xsbS1pbmZlcmVuY2UiLnBvcnRzICs9IFsiODAwMDo4MDAwIl0gf
AogICAgICAgICAgICAgICAuc2VydmljZXMuIm5lbW9sbG0taW5mZXJlbmNlIi5leHBvc2UgKz0g
```

Tabella 3-6. Immagine del container NVIDIA RAG (continua)

Componente	Descrizione
	WyI4MDAwIl0nIHJhZy1hcHAtdGV4dC1jaGF0Ym90Lmpzb24gPiB0ZW1wLmpzb24gJiYgbXY gdGVtcC5qc29uIHJhZy1hcHAtdGV4dC1jaGF0Ym90Lmpzb24KICAgICAgICBweXRob24zIC 1jICJpbXBvcnQgeWFtbCwganNvbiwgc3lzOyBwcmludCh5YW1sLnNhZmVfZHVtcChqc29uL mxvYWQoc3lzLnN0ZGluKSwgZGVmYXVsdF9mbG93X3N0eWxlPUZhbHNlLCBzb3J0X2tleXM9 RmFsc2UpKSIgPCByYWctYXBwLXRleHQtY2hhdGJvdC5qc29uID4gIiR7UkFHX05BTUV9X3Y ke1JBR19WRVJTSU9OfS9yYWctYXBwLXRleHQtY2hhdGJvdC55YW1sIgogICAgICBmaQoKIC AgICAgIyBnZXQgZW1iZWRkaW5nIG1vZGVscwogICAgICBuZ2MgcmVnaXN0cnkgbW9kZWwgZ G93bmxvYWQtdmVyc2lvbiAke0VNQkVEX01PREVMX1VSSX0KICAgICAgY2htb2QgLVIgbyty WCAke0VNQkVEX05BTUV9X3Yke0VNQkVEX1ZFUlNJT059CgogICAgICAjIGNvbmZpZyBjb21 wb3NlLmVudgogICAgICBjYXQgPDwgRU9GID4gY29tcG9zZS5lbnYKICAgICAgZXhwb3J0IE 1PREVMX0RJUkVDVE9SWT0iJHtMTE1fTU9ERUxfRk9MREVSfSIKICAgICAgZXhwb3J0IE1PR EVMX05BTUU9JHtMTE1fTkFNRX0KICAgICAgZXhwb3J0IE5VTV9HUFU9JHtOVU1fR1BVfQog ICAgICBleHBvcnQgQVBQX0NPTkZJR19GSUxFPS9kZXYvbnVsbAogICAgICBleHBvcnQgRU1 CRURESU5HX01PREVMX0RJUkVDVE9SWT0iL29wdC9kYXRhLyR7U1CRURfTkFNRX1fdiR7RU 1CRURfVkVSU0lPTn0iCiAgICAgIGV4cG9ydCBFTUJFRERJTkdfTU9ERUxfTkFNRT0ke0VNQ kVEX1RZUEV9CiAgICAgIGV4cG9ydCBFTUJFRERJTkdfTU9ERUxfQ0tQVF9OQU1FPSIke0VN QkVEX1RZUEV9LSR7RU1CRURfVkVSU0lPTn0ubmVtbyIKICAgIEVPRgoKICAgICAgdG91Y2g gLmZpbGVfZG93bmxvYWRlZAogICAgZmkKCiAgICAgIHN0YXJ0IE5HQyBSQUcKICAgIGRvY2 tlciBjb21wb3NlIC1mICR7UkFHX05BTUV9X3Yke1JBR19WRVJTSU9OfS9kb2NrZXItY29tc G9zZS12ZWN0b3JkYi55YW1sIHVwIC1kIHBndmVjdG9yCiAgICBzb3VyY2UgY29tcG9zZS5l bnY7IGRvY2tlciBjb21wb3NlIC1mICR7UkFHX05BTUV9X3Yke1JBR19WRVJTSU9OfS9yYWc tYXBwLXRleHQtY2hhdGJvdC55YW1sIHVwIC1kCgotIHBhdGg6IC9vcHQvZGx2bS91dGlscy 5zaAogIHBlcm1pc3Npb25zOiAnMDc1NScKICBjb250ZW50OiB8CiAgICAjIS9iaW4vYmFza AogICAgZXJyb3JfZXhpdCgpIHsKICAgICAgZWNobyAiRXJyb3I6ICQxIiA+JjIKICAgICAg dm10b29sc2QgLS1jbWQgImluZm8tc2V0IGd1ZXN0aW5mby52bXNlcnZpY2UuYm9vdHN0cmF wLmNvbmRpdGlvbiBmYWxzZSwgREExXb3JrbG9hZEZhaWx1cmUsICQxIgogICAgICBleGl0ID EKICAgIH0KCiAgICBjaGVja19wcm90b2NvbCgpIHsKICAgICAgbG9jYWwgcHJveHlfdXJsP SQxCiAgICAgIHNoaWZ0CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIp CiAgICAgIGlmIFtbIC1uICIke3Byb3h5X3VybH0iIF1dOyB0aGVuCiAgICAgICAgbG9jYWw gcHJvdG9jb2w9JChlY2hvICIke3Byb3h5X3VybH0iIHwgYXdrIC1GICc6Ly8nICd7aWYgKE 5GID4gMSkgcHJpbnQgJDE7IGVsc2UgcHJpbnQgIiJ9JykKICAgICAgICBpZiBbIC16ICIkc HJvdG9jb2wiIF07IHRoZW4KICAgICAgICAgIGVjaG8gIk5vIHNwZWNpZmljIHByb3RvY29s IHByb3ZpZGVkLiBTa2lwcGluZyBwcm90b2NvbCBjaGVjay4iCiAgICAgICAgICByZXR1cm4 gMAogICAgICAgIGZpCiAgICAgICAgbG9jYWwgcHJvdG9jb2xfaW5jbHVkZWQ9ZmFsc2UKIC AgICAgICBmb3IgdmFyIGluICIke3N1cHBvcnRlZF9wcm90b2NvbHNbQF19IjsgZG8KICAgI CAgICAgIGlmIFtbICIke3Byb3RvY29sfSIgPT0gIiR7dmFyfSIgXV07IHRoZW4KICAgICAg ICAgICAgcHJvdG9jb2xfaW5jbHVkZWQ9dHJ1ZQogICAgICAgICAgICBicmVhawogICAgICA gICAgZmkKICAgICAgICBkb25lCiAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2xfaW5jbHVkZW R9IiA9PSBmYWxzZSBdXTsgdGhlbgogICAgICAgICAgZXJyb3JfZXhpdCAiVW5zdXBwb3J0Z WQgcHJvdG9jb2w6ICR7cHJvdG9jb2x9LiBTdXBwb3J0ZWQgcHJvdG9jb2xzIGFyZTogJHtz dXBwb3J0ZWRfcHJvdG9jb2xzWypdfSIKICAgICAgICAgICBmaQogICAgICBmaQogICAgfQoKICA gICMgJEA6IGxpc3Qgb2Ygc3VwcG9ydGVkIHByb3RvY29scwogICAgc2V0X3Byb3h5KCkgew ogICAgICBsb2NhbCBzdXBwb3J0ZWRfcHJvdG9jb2xzPSgiJEAiKQoKICAgICAgQ09ORklHX 0pTT05fQkFTRTY0PSQoZ3JlcCAnY29uZmlnLWpzb24nIC9vcHQvZGx2bS9vdmYtZW52Lnht bCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKICAgICAgQ09 ORklHX0pTT049JChlY2hvICR7Q09ORklHX0pTT05fQkFTRTY0fSB8IGJhc2U2NCAtLWRlY2 9kZSkKCiAgICAgIEhUVFBfUFJPWFlfVVJMPSQoZWNobyAiJHtDT05GSUdfSlNPTn0iIHwga nEgLXIgJy5odHRwX3Byb3h5IC8vIGVtcHR5JykKICAgICAgSFRUUFNfUFJPWFlfVVJMPSQo ZWNobyAiJHtDT05GSUdfSlNPTn0iIHwganEgLXIgJy5odHRwc19wcm94eSAvLyBlbXB0eSc pCiAgICAgIGlmIFtbICQ/ IC1uZSAwIHx8ICgteiAiJHtIVFRQX1BST1hZX1VSTH0iICYmIC16ICIke0hUVFBTX1BST1h ZX1VSTH0iKSBdXTsgdGhlbgogICAgICAgIGluZm86IFRoZSBjb25maWctctanNvbi B3YXMgcGFyc2VkLCBidXQgbm8gcHJveHkgc2V0dGluZ3Mgd2VyZSBmb3VuZC4iCiAgICAgI CAgcmV0dXJuIDAKICAgICAgZmkKCiAgICAgIGNoZWNrX3Byb3RvY29sICIke0hUVFBfUFJP WFlfVVJMfSIgIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iCiAgICAgIGNoZWNrX3Byb3R vY29sICIke0hUVFBTX1BST1hZX1VSTH0iICIke3N1cHBvcnRlZF9wcm90b2NvbHNbQF19Ig oKICAgICAgaWYgISBncmVwIC1xICdodHRwX3Byb3h5JyAvZXRjL2Vudmlyb25tZW50OyB0a GVuCiAgICAgICAgZWNobyAiZXhwb3J0IGh0dHBfcHJveHk9JHtIVFRQX1BST1hZX1VSTH0K ICAgICAgICBleHBvcnQgaHR0cHNfcHJveHk9JHtIVFRQU19QUk9YWV9VUkx9CiAgICAgICA gZXhwb3J0IEhUVFBfUFJPWFk9JHtIVFRQX1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSF RUUFNfUFJPWFk9JHtIVFRQU19QUk9YWV9VUkx9CiAgICAgICAgZXhwb3J0IG5vX3Byb3h5P

Tabella 3-6. Immagine del container NVIDIA RAG (continua)

Componente	Descrizione


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


che corrisponde allo script seguente in formato testo normale:

```
#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
```

Tabella 3-6. Immagine del container NVIDIA RAG (continua)

Componente	Descrizione

```
      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}
```

Tabella 3-6. Immagine del container NVIDIA RAG (continua)

Componente	Descrizione

```
      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$
```

Tabella 3-6. Immagine del container NVIDIA RAG (continua)

Componente	Descrizione

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

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

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

      touch .file_downloaded
    fi

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

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

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

Tabella 3-6. Immagine del container NVIDIA RAG (continua)

Componente	Descrizione
	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" } ■ Immettere le proprietà di installazione del driver guest vGPU, ad esempio `vgpu-license` e `nvidia-portal-api-key`. ■ Specificare i valori per le proprietà necessarie per un ambiente disconnesso in base alle esigenze. Vedere Proprietà OVF delle macchine virtuali di deep learning.
Output	■ Registri di installazione per il driver guest della vGPU in `/var/log/vgpu-install.log`. Per verificare che il driver guest della vGPU sia installato, accedere alla macchina virtuale tramite SSH ed eseguire il comando `nvidia-smi`.

Tabella 3-6. Immagine del container NVIDIA RAG (continua)

Componente	Descrizione
	■ Registri dello script cloud-init in `/var/log/dl.log`. Per tenere traccia dello stato di avanzamento della distribuzione, eseguire `tail -f /var/log/dl.log` . ■ Applicazione Web del chatbot di esempio a cui è possibile accedere all'indirizzo `http://dl_vm_ip:3001/orgs/nvidia/models/text-qa-chatbot` È possibile caricare la propria knowledge base.

Assegnazione di un indirizzo IP statico a una macchina virtuale di deep learning in VMware Private AI Foundation with NVIDIA

Per impostazione predefinita, le immagini della macchina virtuale di deep learning sono configurate con l'assegnazione dell'indirizzo DHCP. Se si desidera distribuire una macchina virtuale di deep learning con un indirizzo IP statico direttamente in un cluster vSphere, è necessario aggiungere ulteriore codice alla sezione cloud-init.

In vSphere with Tanzu, l'assegnazione dell'indirizzo IP è determinata dalla configurazione di rete per il supervisore in NSX.

Procedura

1 Creare uno script cloud-init in formato testo normale per il carico di lavoro DL che si intende utilizzare.

Vedere Carichi di lavoro di deep learning in VMware Private AI Foundation with NVIDIA.

2 Inserire il codice aggiuntivo seguente nello 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 Codificare lo script cloud-init risultante in formato base64.

4 Impostare lo script cloud-init risultante in formato base64 come valore per il parametro OVF `user-data` dell'immagine della macchina virtuale di deep learning.

Esempio: Assegnazione di un indirizzo IP statico a un carico di lavoro di esempio CUDA

Per una macchina virtuale di deep learning di esempio con un carico di lavoro DL di Carichi di lavoro di deep learning in VMware Private AI Foundation with NVIDIA:

Elemento macchina virtuale di deep learning	Valore di esempio
Immagine del carico di lavoro DL	nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8
Indirizzo IP	10.199.118.245
Prefisso subnet	/25
Gateway	10.199.118.253
Server DNS	■ 10.142.7.1 ■ 10.132.7.1

si specifica il codice cloud-init seguente:

```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBwLnNoCiAgcGVybWlzc2lvbnM6IC
cwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBkb2NrZXIgcnVuIC1kIG52Y3IuaW8vbnZpZGlhL2s4
cy9jdWRhLXNhbXBsZTp2ZWN0b3JhZGQtY3VkYTExLjcuMS11Ymk4CgptYW5hZ2VfZXRjX2hvc3RzOiB0cnVlCiAKd3JpdG
VfZmlsZXM6CiAgLSBwYXRoOiAvZXRjL25ldHBsYW4vNTAtY2xvdWQtaW5pdC55YW1sCiAgICBwZXJtaXNzaW9uczogJzA2
MDAnCiAgICBjb250ZW50OiB8CiAgICAgIG5ldHdvcms6CiAgICAgICAgdmVyc2lvbjogMgogICAgICAgIHJlbmRlcmVyOi
BuZXR3b3JrZAogICAgICAgIGV0aGVybmV0czoKICAgICAgICAgIGVuczMzOgogICAgICAgICAgICBkaGNwNDogZmFsc2Ug
IyBkaXNhYmxlIERIQ1A0CiAgICAgICAgICAgIGFkZHJlc3NlczogWzEwLjE5OS4xMTguMjQ1LzI1XSAgIyBTZXQgdGhlIH
N0YXRpYyBJUCBhZGRyZXNzIGFuZCBtYXNrCiAgICAgICAgICAgIHJvdXRlczoKICAgICAgICAgICAgICAgIC0gdG86IGRl
ZmF1bHQKICAgICAgICAgICAgICAgIHZpYTogMTE4LjI1MyAjIENvbmZpZ3VyZSBnYXRld2F5CiAgICAgICAgICAgICAgIC
AgICAgICAgICAgIG5hbWVzZXJ2ZXJzOgogICAgICAgICAgICAgICAgICAgIGFkZHJlc3NlczogWzEwLjE0Mi43LjEsIDEwLjEzMi43LjFdICMg
UHJvdmlkZSB0aGUgRE5TIHNlcnZlciBhZGRyZXNzLiBTZXBhcmF0ZSBtdWxpdHBsZSBETlMgc2VydmVyIGFkZHJlc3Nlcy
B3aXRoIGNvbW1hcy4KIApydW5jbWQ6CiAgLSBuZXRwbGFuIGFwcGx5
```

che corrisponde allo script seguente in formato testo normale:

```
#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
```

Configurazione di un'stanza di Deep Learning VM con un server proxy

Per connettere l'istanza di Deep Learning VM a Internet in un ambiente disconnesso in cui l'accesso a Internet viene eseguito tramite un server proxy, è necessario specificare i dettagli del server proxy nel file `config.json` nella macchina virtuale.

Procedura

1 Creare un file JSON con le proprietà per il server proxy.

Server proxy che non richiede l'autenticazione	`{` `  "http_proxy": "protocol://ip-address-or-fqdn:port",` `  "https_proxy": "protocol://ip-address-or-fqdn:port"` `}`
Server proxy che richiede l'autenticazione	`{` `  "http_proxy": "protocol://username:password@ip-address-or-fqdn:port",` `  "https_proxy": "protocol://username:password@ip-address-or-fqdn:port"` `}`

dove:

- *protocol* è il protocollo di comunicazione utilizzato dal server proxy, ad esempio `http` o `https`.

- *username* e *password* sono le credenziali per l'autenticazione nel server proxy. Se il server proxy non richiede l'autenticazione, ignorare questi parametri.
- *ip-address-or-fqdn*: indirizzo IP o nome host del server proxy.
- *port*: numero della porta in cui il server proxy è in ascolto delle richieste in arrivo.

2 Codificare il codice JSON risultante in formato base64.

3 Quando si distribuisce l'immagine di Deep Learning VM, aggiungere il valore codificato alla proprietà OVF `config-json`.

Risoluzione dei problemi relativi alla distribuzione di un'istanza di Deep Learning VM in VMware Private AI Foundation with NVIDIA

Le informazioni sulla risoluzione dei problemi relativi alla distribuzione di Deep Learning VM in VMware Private AI Foundation with NVIDIA forniscono soluzioni ai potenziali problemi che potrebbero verificarsi.

- L'automazione del carico di lavoro DL non viene eseguita

  Dopo aver distribuito Deep Learning VM in VMware Private AI Foundation with NVIDIA, il carico di lavoro DL specificato non è in esecuzione.

- Il download di un carico di lavoro DL non riesce perché le credenziali di autenticazione non sono valide

  Dopo aver distribuito Deep Learning VM in VMware Private AI Foundation with NVIDIA, il download del carico di lavoro DL specificato nella macchina virtuale non riesce e nel file di log vengono visualizzati messaggi di errore che indicano che le credenziali di autenticazione non sono valide.

- Il download del driver guest NVIDIA vGPU non riesce perché manca un collegamento di download

  Dopo aver distribuito Deep Learning VM, il download del driver guest della vGPU specificato nella macchina virtuale non riesce e vengono visualizzati messaggi di errore che indicano che manca un collegamento o una risorsa di download.

- Lo stato del driver guest NVIDIA vGPU è Non concesso in licenza

  Dopo aver distribuito un'istanza di Deep Learning VM in VMware Private AI Foundation with NVIDIA, lo stato del driver guest NVIDIA vGPU è Non concesso in licenza.

L'automazione del carico di lavoro DL non viene eseguita

Dopo aver distribuito Deep Learning VM in VMware Private AI Foundation with NVIDIA, il carico di lavoro DL specificato non è in esecuzione.

Problema

Si distribuisce Deep Learning VM con un carico di lavoro DL da preinstallare all'avvio iniziale. Dopo aver avviato Deep Learning VM, il carico di lavoro DL non viene eseguito.

Causa

1 Il valore `user-data` o i valori di altri parametri OVF con codifica base64, come `image-oneliner` o `config-json`, vengono salvati o decodificati in modo errato nel file `/opt/dlvm/dl_app.sh`. Di conseguenza, lo script del carico di lavoro DL non viene eseguito.

2 L'installazione del driver vGPU non riesce causando la mancata esecuzione dello script cloud-init passato al parametro OVF `user-data`. Lo script cloud-init si basa sulla corretta installazione del driver NVIDIA vGPU.

Soluzione

In Deep Learning VM, verificare se il carico di lavoro DL è installato nella macchina virtuale e applicare una soluzione di conseguenza.

Disponibilità del carico di lavoro DL	Soluzione
I componenti del carico di lavoro DL non vengono creati nella macchina virtuale.	■ Se si utilizza uno script cloud-init come input per il parametro OVF `user-data`, verificare i valori seguenti: ■ Controllare lo script codificato e immesso come input `user-data`. Assicurarsi che il valore `#cloud-config` sia presente nella prima riga e che sia incluso nell'equivalente base64. ■ Controllare il parametro `path`. ■ Controllare la stringa con codifica base64 e assicurarsi che il valore `user-data` sia salvato correttamente in `/opt/dlvm/dl_app.sh`. ■ Se si utilizzano altri parametri OVF, verificare i valori seguenti: ■ `image-oneliner`. Controllare la stringa con codifica base64 e assicurarsi che il comando a una riga sia salvato correttamente in `/opt/dlvm/dl_app.sh`. ■ `config-json`. Controllare la stringa con codifica base64 e assicurarsi che il file di composizione Docker e `config.json`, se specificati, siano salvati correttamente in `/root/docker/compose.yaml` e `/root/.docker/config.json`. Per informazioni sui parametri OVF dell'immagine di Deep Learning VM più recente, vedere Proprietà OVF delle macchine virtuali di deep learning.
I componenti del carico di lavoro DL vengono creati ma il carico di lavoro non è in esecuzione.	■ Controllare i messaggi di errore in `/var/log/vgpu-install.log`. ■ Se si utilizza uno script cloud-init come input per il parametro OVF `user-data`, verificare che il driver NVIDIA vGPU sia installato e funzioni correttamente. Lo script cloud-init non viene eseguito se l'installazione del driver NVIDIA vGPU non riesce.

Il download di un carico di lavoro DL non riesce perché le credenziali di autenticazione non sono valide

Dopo aver distribuito Deep Learning VM in VMware Private AI Foundation with NVIDIA, il download del carico di lavoro DL specificato nella macchina virtuale non riesce e nel file di log vengono visualizzati messaggi di errore che indicano che le credenziali di autenticazione non sono valide.

Problema

Se si installa un'immagine del container del carico di lavoro DL, ad esempio Triton Inference Server, TensorFlow o Pytorch, il file `/var/log/dl.log` contiene il messaggio seguente:

```
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>
```

Per NVIDIA RAG, il file `/var/log/dl.log` contiene il messaggio seguente:

```
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
```

Causa

L'autenticazione nel registro del container nvcr.io non è riuscita. Di conseguenza, l'immagine del carico di lavoro DL non può essere scaricata nella macchina virtuale.

Soluzione

- Verificare le credenziali per l'accesso al registro nvcr.io passate come parametri OVF o alla procedura guidata di configurazione del catalogo per Private AI in VMware Aria Automation.
  - Registro: nvcr.io
  - Account utente del registro: $oauthtoken
  - Password del registro: *NGC portal API key*
- Verificare che la chiave API del portale NVIDIA NGC disponga delle autorizzazioni per accedere alle risorse necessarie e che la chiave non sia scaduta.

Il download del driver guest NVIDIA vGPU non riesce perché manca un collegamento di download

Dopo aver distribuito Deep Learning VM, il download del driver guest della vGPU specificato nella macchina virtuale non riesce e vengono visualizzati messaggi di errore che indicano che manca un collegamento o una risorsa di download.

Problema

Il file `/var/log/vgpu-install.log` contiene uno dei messaggi seguenti:

```
Error No download link detected via API
```

```
No downloads found via API
```

Causa

La chiave API del portale delle licenze NVIDIA passata come valore alla proprietà OVF `nvidia-portal-api-key` o alla configurazione guidata del catalogo per Private AI in VMware Aria Automation non è valida, è scaduta o è formattata in modo errato.

Soluzione

- Verificare che la chiave API sia valida.

- Verificare che la chiave API sia stata immessa correttamente.

La chiave API utilizza in genere il formato UUID versione 4 `xxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

Lo stato del driver guest NVIDIA vGPU è Non concesso in licenza

Dopo aver distribuito un'istanza di Deep Learning VM in VMware Private AI Foundation with NVIDIA, lo stato del driver guest NVIDIA vGPU è Non concesso in licenza.

Problema

Il file `/var/log/vgpu-install.log` contiene uno dei messaggi seguenti:

```
License Status: Unlicensed
```

```
Unlicensed (Restricted)
```

Causa

Il token di configurazione del client NVIDIA vGPU passato come valore alla proprietà OVF `vgpu-license` o alla procedura guidata di configurazione del catalogo per Private AI in VMware Aria Automation non è valido, è scaduto o è formattato in modo errato.

Soluzione

- Verificare la validità del token di configurazione del client.
- Verificare che la licenza di vGPU sia formattata correttamente e che segua il formato del token JWT, che in genere ha l'aspetto `eyxxxx.eyxxxxx.xxxxx`.

  È possibile decrittografare il token JWT in jwt.io per controllare la data di scadenza e l'URL del server del nodo.
- Il token di licenza di vGPU è stato salvato anche in `/etc/nvidia/ClientConfigToken/client_configuration_token.tok`.
- Per risolvere ulteriormente il problema, eseguire questo comando per verificare la presenza di messaggi di errore specifici relativi alla comunicazione con il server delle licenze NVIDIA.

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

Per applicare un nuovo token, eseguire i passaggi seguenti:

1 Sostituire il contenuto del file `/etc/nvidia/ClientConfigToken/client_configuration_token.tok` con un nuovo token ed eseguire il comando seguente:

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

2 Riavviare il servizio NVIDIA.

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

3 Verificare lo stato della licenza del driver guest NVIDIA vGPU.

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

Distribuzione dei carichi di lavoro AI nei cluster TKG in VMware Private AI Foundation with NVIDIA

4

In qualità di tecnico DevOps, è possibile distribuire carichi di lavoro AI dei container in cluster TKG (Tanzu Kubernetes Grid) i cui nodi worker sono accelerati con GPU NVIDIA.

Per informazioni sul supporto dei carichi di lavoro AI nei cluster TKG, vedere Informazioni sulla distribuzione di carichi di lavoro AI/ML nei cluster TKGS.

Leggi i seguenti argomenti:

- Provisioning di un cluster TKG con accelerazione GPU tramite un catalogo self-service in VMware Private AI Foundation with NVIDIA
- Provisioning di un cluster TKG con accelerazione GPU tramite il comando `kubectl` in un ambiente di VMware Private AI Foundation with NVIDIA connesso
- Provisioning di un cluster TKG con accelerazione GPU tramite il comando `kubectl` in un ambiente di VMware Private AI Foundation with NVIDIA disconnesso

Provisioning di un cluster TKG con accelerazione GPU tramite un catalogo self-service in VMware Private AI Foundation with NVIDIA

In VMware Private AI Foundation with NVIDIA in qualità di tecnico DevOps, è possibile eseguire il provisioning di un cluster TKG accelerato con GPU NVIDIA da VMware Aria Automation utilizzando gli elementi catalogo self-service di un cluster Kubernetes AI in Automation Service Broker. È quindi possibile distribuire le immagini dei container AI da NVIDIA NGC nel cluster.

Prerequisiti

Verificare con l'amministratore del cloud che VMware Private AI Foundation with NVIDIA sia configurato. Vedere Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI.

Procedura

- In Automation Service Broker distribuire un elemento catalogo di un cluster Kubernetes AI nell'istanza del supervisore configurata dall'amministratore del cloud.
  - Per un cluster Tanzu Kubernetes Grid non RAG, utilizzare l'elemento catalogo **Cluster Kubernetes AI**. Vedere Distribuzione di un cluster Tanzu Kubernetes Grid con accelerazione GPU.
  - Per un cluster Tanzu Kubernetes Grid basato su RAG, utilizzare l'elemento catalogo **Cluster RAG Kubernetes AI**. Vedere Distribuzione di un cluster RAG Tanzu Kubernetes Grid con accelerazione GPU.

Operazioni successive

Eseguire un'immagine del container AI. In un ambiente connesso, utilizzare il catalogo NVIDIA NGC. In un ambiente disconnesso, utilizzare il registro Harbor nel supervisore.

Per un cluster Tanzu Kubernetes Grid basato su RAG, distribuire un database PostgreSQL pgvector in VMware Data Services Manager e installare la pipeline di esempio di RAG da NVIDIA. Vedere Distribuzione di un carico di lavoro RAG in un cluster TKG.

Provisioning di un cluster TKG con accelerazione GPU tramite il comando `kubectl` in un ambiente di VMware Private AI Foundation with NVIDIA connesso

In VMware Private AI Foundation with NVIDIA in qualità di tecnico DevOps, utilizzando l'API Kubernetes, eseguire il provisioning di un cluster TKG che utilizza GPU NVIDIA. È quindi possibile distribuire carichi di lavoro AI dei container dal catalogo NVIDIA NGC.

Utilizzare `kubectl` per distribuire il cluster TKG nello spazio dei nomi configurato dall'amministratore del cloud.

Prerequisiti

Verificare con l'amministratore del cloud che siano soddisfatti i prerequisiti seguenti per l'infrastruttura pronta per AI.

- VMware Private AI Foundation with NVIDIA è configurato. Vedere Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI.
- In un ambiente disconnesso, una libreria di contenuti con immagini TKr Ubuntu deve essere aggiunta allo spazio dei nomi vSphere per i carichi di lavoro AI. Vedere Configurazione di una libreria di contenuti con TKr Ubuntu per un ambiente di VMware Private AI Foundation with NVIDIA disconnesso.

Procedura

1 Accedere al piano di controllo del supervisore.

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

2 Eseguire il provisioning di un cluster TKG e installare NVIDIA GPU Operator e NVIDIA Network Operator in tale cluster.

Vedere Workflow dell'operatore del cluster per la distribuzione di carichi di lavoro AI/ML nei cluster TKGS.

Operazioni successive

Distribuire un'immagine del container AI dal catalogo NVIDIA NGC.

Provisioning di un cluster TKG con accelerazione GPU tramite il comando `kubectl` in un ambiente di VMware Private AI Foundation with NVIDIA disconnesso

In VMware Private AI Foundation with NVIDIA in qualità di tecnico DevOps, utilizzando l'API Kubernetes, eseguire il provisioning di un cluster TKG che utilizza GPU NVIDIA. In un ambiente disconnesso, è necessario configurare anche un repository di pacchetti Ubuntu locale e utilizzare il registro Harbor per il supervisore.

Prerequisiti

Verificare con l'amministratore del cloud che siano soddisfatti i prerequisiti seguenti per l'infrastruttura pronta per AI.

- VMware Private AI Foundation with NVIDIA è configurato per un ambiente disconnesso. Vedere Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI.
- Una macchina che ha accesso all'endpoint supervisore e al repository Helm locale che ospita le definizioni del grafico di NVIDIA GPU Operator.

Procedura

1 Eseguire il provisioning di un cluster TKG nello spazio dei nomi vSphere configurato dall'amministratore del cloud.

Vedere Provisioning di un cluster TKGS per NVIDIA vGPU.

2 Installare NVIDIA GPU Operator.

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

3 Monitorare l'operazione.

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

Passaggi successivi

Distribuire un'immagine del container AI dal registro Harbor al supervisore.

Distribuzione di carichi di lavoro RAG in VMware Private AI Foundation with NVIDIA

5

Un carico di lavoro RAG (Retrieval-Augmented Generation) è costituito da un LLM e da una knowledge base esterna con i dati più recenti, archiviati in un database vettore. In VMware Private AI Foundation with NVIDIA è possibile configurare un carico di lavoro RAG per utilizzare le integrazioni di un database vettore gestito da VMware Data Services Manager.

Leggi i seguenti argomenti:

- Distribuzione di un database vettore in VMware Private AI Foundation with NVIDIA
- Distribuzione di una macchina virtuale di deep learning con un carico di lavoro RAG
- Distribuzione di un carico di lavoro RAG in un cluster TKG

Distribuzione di un database vettore in VMware Private AI Foundation with NVIDIA

Se si prevede di utilizzare Retrieval-Augmented Generation (RAG) con VMware Private AI Foundation with NVIDIA, configurare un database PostgreSQL con pgvector utilizzando VMware Data Services Manager.

È possibile creare il database manualmente oppure creare un catalogo self-service in VMware Aria Automation che possa essere utilizzato dagli sviluppatori e dai tecnici DevOps.

Prerequisiti

- Verificare che VMware Private AI Foundation with NVIDIA sia disponibile per il dominio del carico di lavoro VI. Vedere Distribuzione di VMware Private AI Foundation with NVIDIA.
- Verificare con l'amministratore del cloud che i prerequisiti per la creazione di un database PostgreSQL siano soddisfatti. Vedere Creazione di database.
- Installare l'utilità della riga di comando `psql` dal sito Web di PostgreSQL.

Procedura

1 Distribuire un database PostgreSQL nel dominio del carico di lavoro VI e recuperare la stringa di connessione per il database.

È possibile utilizzare uno dei workflow seguenti. Se si è un data scientist, è possibile distribuire direttamente un database da VMware Aria Automation. In caso contrario, richiedere una distribuzione del database all'amministratore DSM o all'utente DSM.

Workflow di distribuzione	Ruolo utente obbligatorio	Descrizione
Distribuire e recuperare la stringa di connessione di un database PostgreSQL da VMware Aria Automation.	Data scientist o tecnico DevOps	Vedere Distribuzione di un database vettore mediante un elemento catalogo self-service in VMware Aria Automation.
Distribuire e recuperare la stringa di connessione di un database PostgreSQL dalla console di VMware Data Services Manager.	L'amministratore DSM o l'utente DSM oppure un amministratore del cloud ha assegnato uno di questi ruoli	Vedere Creazione di database e Connessione a un database.
Distribuire e recuperare la stringa di connessione di un database PostgreSQL utilizzando il comando `kubectl`.	L'amministratore DSM o l'utente DSM o un tecnico DevOps ha assegnato uno di questi ruoli	Vedere Abilitazione dell'utilizzo self-service di VMware Data Services Manager.

Il formato della stringa di connessione del database distribuito è il seguente.

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

2 Attivare l'estensione pgvector nel database mediante l'utilità della riga di comando `psql`.

a Connettersi al database.

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

b Attivare l'estensione pgvector.

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

Operazioni successive

Integrare il database nel carico di lavoro RAG. Vedere Distribuzione di una macchina virtuale di deep learning con un carico di lavoro RAG e Distribuzione di un carico di lavoro RAG in un cluster TKG.

Distribuzione di un database vettore mediante un elemento catalogo self-service in VMware Aria Automation

In VMware Private AI Foundation with NVIDIA in qualità di data scientist o tecnico DevOps, è possibile distribuire un database vettore da VMware Aria Automation utilizzando un elemento catalogo self-service in Automation Service Broker.

Procedura

1. Accedere a VMware Aria Automation e, in Automation Service Broker, individuare l'elemento catalogo per la distribuzione del database in base alle informazioni ricevute dall'amministratore del cloud.

   Per impostazione predefinita, l'elemento catalogo è denominato **DSM DBaaS**.

2. Nella scheda dell'elemento catalogo fare clic su **Richiedi** e immettere i dettagli del nuovo database PostgreSQL.

   Per ulteriori informazioni sulle impostazioni del database, vedere Creazione di database.

3. Recuperare la stringa di connessione del database distribuito.
   a. In Automation Service Broker fare clic su **Distribuzioni > Distribuzioni**.
   b. Selezionare la voce di distribuzione per il database.
   c. Nella scheda **Topologia** selezionare il modello cloud per la distribuzione del database e dal menu **Azioni** per il modello selezionare **Ottieni stringa di connessione**.

Risultati

Per ulteriori informazioni sul provisioning e l'esecuzione di operazioni relative ai database in VMware Data Services Manager da VMware Aria Automation, vedere il file `readme.md` nel bundle `AriaAutomation_DataServicesManager`.

Distribuzione di una macchina virtuale di deep learning con un carico di lavoro RAG

È possibile distribuire un'istanza di Deep Learning VM con un carico di lavoro NVIDIA RAG utilizzando un database PostgreSQL pgvector gestito da VMware Data Services Manager.

Per informazioni sul carico di lavoro NVIDIA RAG, vedere la documentazione NVIDIA RAG Applications Docker Compose (richiede autorizzazioni specifiche dell'account).

Prerequisiti

- Verificare che VMware Private AI Foundation with NVIDIA sia configurato. Vedere Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI.
- Distribuzione di un database vettore in VMware Private AI Foundation with NVIDIA.

Procedura

1 Se in qualità di data scientist si distribuisce Deep Learning VM utilizzando un elemento catalogo in VMware Aria Automation, specificare i dettagli del database PostgreSQL pgvector dopo aver distribuito la macchina virtuale.

   a Distribuzione di una workstation RAG in VMware Aria Automation.

   b Passare a **Utilizza > Distribuzioni > Distribuzioni** e individuare la distribuzione della macchina virtuale di deep learning.

   c Nella sezione **Macchina virtuale workstation** salvare i dettagli per l'accesso SSH alla macchina virtuale.

   d Accedere alla macchina virtuale di deep learning tramite SSH utilizzando le credenziali disponibili in Automation Service Broker.

   e Aggiungere le variabili di pgvector seguenti nel file `/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 Riavviare l'applicazione multi-container NVIDIA RAG eseguendo i comandi seguenti.

   Ad esempio, per 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 Se in qualità di tecnico DevOps si distribuisce Deep Learning VM per un data scientist direttamente nel cluster vSphere o utilizzando il comando `kubectl`, creare uno script cloud-init e distribuire Deep Learning VM.

   a Creare uno script cloud-init per NVIDIA RAG e il database PostgreSQL pgvector creato.

   È possibile modificare la versione iniziale dello script cloud-init per NVIDIA RAG. Ad esempio, per NVIDIA RAG 24.03 e un database PostgreSQL pgvector con dettagli di connessione `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 Codificare lo script cloud-init nel formato base64.

Utilizzare uno strumento per la codifica base 64, ad esempio https://decode64base.com/ per generare la versione codificata dello script cloud-init.

c Distribuire la macchina virtuale di deep learning passando il valore base64 dello script cloud-init al parametro di input `user-data`.

Vedere Distribuzione di una macchina virtuale di deep learning direttamente in un cluster vSphere in VMware Private AI Foundation with NVIDIA o Distribuzione di una macchina virtuale di deep learning tramite il comando kubectl in VMware Private AI Foundation with NVIDIA.

Distribuzione di un carico di lavoro RAG in un cluster TKG

In qualità di tecnico DevOps, in un cluster TKG in un supervisore è possibile distribuire un carico di lavoro RAG basato sulla pipeline di esempio di RAG di NVIDIA che utilizza un database PostgreSQL pgvector gestito da VMware Data Services Manager.

Prerequisiti

- Verificare che VMware Private AI Foundation with NVIDIA sia disponibile per il dominio del carico di lavoro VI. Vedere Capitolo 2 Preparazione di VMware Cloud Foundation per la distribuzione del carico di lavoro di Private AI.
- Distribuzione di un database vettore in VMware Private AI Foundation with NVIDIA.

Procedura

1 Eseguire il provisioning di un cluster TKG con accelerazione GPU.

È possibile utilizzare uno dei workflow seguenti.

Workflow di provisioning	Passaggi
Utilizzando un elemento catalogo in VMware Aria Automation	Distribuzione di un cluster RAG Tanzu Kubernetes Grid con accelerazione GPU.
Utilizzando il comando `kubectl`	1 Eseguire il provisioning di un cluster TKG con accelerazione GPU utilizzando il comando `kubectl`. ■ Per un ambiente connesso, vedere Provisioning di un cluster TKG con accelerazione GPU tramite il comando kubectl in un ambiente di VMware Private AI Foundation with NVIDIA connesso. ■ Per un ambiente disconnesso, vedere Provisioning di un cluster TKG con accelerazione GPU tramite il comando kubectl in un ambiente di VMware Private AI Foundation with NVIDIA disconnesso. 2 Installare l'operatore RAG LLM. Vedere Installazione dell'operatore RAG LLM.

2 Se si utilizza il comando `kubectl` per eseguire il provisioning del cluster TKG, installare l'operatore NVIDIA RAG LLM nel cluster TKG.

   Vedere Installazione dell'operatore RAG LLM.

   Durante la distribuzione, l'elemento catalogo **Cluster RAG Kubernetes AI** in VMware Aria Automation installa automaticamente l'operatore NVIDIA RAG LLM nel cluster TKG.

3 Scaricare i manifesti per la pipeline RAG di esempio di NVIDIA.

   Vedere Pipeline RAG di esempio.

4 Configurare la pipeline RAG di esempio con il database PostgreSQL pgvector.

   a Modificare il file YAML della pipeline di esempio.

   Vedere il passaggio 4 in Pipeline RAG di esempio.

   b Nel file YAML configurare la pipeline di esempio con il database PostgreSQL pgvector utilizzando la stringa di connessione del database.

   Vedere Database vettore per la pipeline di esempio RAG.

5 Per fornire un IP esterno per l'applicazione della chat di esempio, nel file YAML impostare `frontend.service.type` su `loadBalancer`.

6 Avviare la pipeline RAG di esempio.

   Vedere Pipeline RAG di esempio.

7 Per accedere all'applicazione di chat di esempio, eseguire il comando seguente per ottenere l'indirizzo IP esterno dell'applicazione.

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

8 In un browser Web, aprire l'applicazione di chat di esempio all'indirizzo **`http://application_external_ip:3001/orgs/nvidia/models/text-qa-chatbot`**.

Monitoraggio di VMware Private AI Foundation with NVIDIA

6

È possibile monitorare le metriche di GPU a livello di cluster e host in vSphere Client e VMware Aria Operations.

In VMware Aria Operations è possibile monitorare le metriche di GPU a livello di cluster, di sistema host e di proprietà dell'host. Per ulteriori informazioni, vedere Dashboard di Private AI (GPU) e Proprietà dei componenti di vCenter Server in VMware Aria Operations.

In vSphere Client è possibile monitorare le metriche di GPU nel modo seguente:

- A livello di host. Vedere Grafici delle prestazioni degli host in vSphere.
- A livello di cluster nei grafici personalizzati. Vedere Utilizzo di grafici avanzati e personalizzati in vSphere.