VMware Private AI Foundation with NVIDIA Guide

23 JUL 2024

VMware Cloud Foundation 5.2

You can find the most up-to-date technical documentation on the VMware by Broadcom website at:

https://docs.vmware.com/

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

Contents

About the VMware Private AI Foundation with NVIDIA Guide

The *VMware Private AI Foundation with NVIDIA Guide* provides an overview of the components of VMware Private AI Foundation with NVIDIA and high-level workflows for development and production use cases.

Intended Audience

The information in *VMware Private AI Foundation with NVIDIA Guide* is intended for data center cloud administrators, data scientists, and DevOps engineers who are familiar with:

- Cloud administrators
  - Concepts of virtualization and software-defined data centers (SDDCs)
  - Hardware components such as top-of-rack (ToR) switches, inter-rack switches, servers with direct attached storage, cables, and power supplies
  - Methods for setting up NVIDIA GPUs on servers in a data center
  - Using VMware vSphere® to work with virtual machines.
  - Using vSphere IaaS control plane to configure and assign vSphere resources to vSphere namespaces on a Supervisor.

  As a cloud administrator, see the following information:

  - Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment
  - Chapter 3 Deploying a Deep Learning VM in VMware Private AI Foundation with NVIDIA
  - Chapter 6 Monitoring VMware Private AI Foundation with NVIDIA
- Data scientists
  - Containers, including Docker, Helm charts and Harbor Registry

  As a data scientist, see the following information:

  - Chapter 3 Deploying a Deep Learning VM in VMware Private AI Foundation with NVIDIA
  - Chapter 5 Deploying RAG Workloads in VMware Private AI Foundation with NVIDIA
- DevOps engineers
  - Provisioning virtual machines in vSphere using the Kubernetes API.
  - Containers, including Docker, Helm charts and Harbor Registry

- Working with vSphere IaaS control plane for provisioning VMs and Tanzu Kubernetes Grid (TKG) clusters.

As a DevOps engineer, see the following information:

- Chapter 4 Deploying AI Workloads on TKG Clusters in VMware Private AI Foundation with NVIDIA
- Chapter 5 Deploying RAG Workloads in VMware Private AI Foundation with NVIDIA

VMware Software Components

The functionality of the VMware Private AI Foundation with NVIDIA solution is available across several software components according to your role in your organization.

Target User Role	Software Category	Supported Software Versions
Cloud administrators	Components that are deployed in VMware Cloud Foundation	See VMware Components in VMware Private AI Foundation with NVIDIA.
Data scientists	Deep learning VM components	See VMware Deep Learning VM Release Notes.
DevOps engineers	TK releases (TKr)	See VMware Tanzu Kuberenetes releases Release Notes.

Related VMware Documentation

The VMware Private AI Foundation with NVIDIA solution includes a stack of VMware software products and components. The documentation for those software products is as follows:

- VMware Cloud Foundation Documentation
- VMware vSphere and vSAN Documentation
- VMware vSphere IaaS Control Plane Documentation
- VMware Aria Automation Documentation
- VMware Aria Operations Documentation
- VMware Aria Suite Lifeycle Documentation
- VMware Data Services Manager Documentation

VMware Cloud Foundation Glossary

The VMware Cloud Foundation Glossary defines terms specific to VMware Cloud Foundation.

What is VMware Private AI Foundation with NVIDIA?

1

As a multi-component solution, you can use VMware Private AI Foundation with NVIDIA to run generative AI workloads by using accelerated computing from NVIDIA, and virtual infrastructure management and cloud management from VMware Cloud Foundation.

VMware Private AI Foundation with NVIDIA provides a platform for provisioning AI workloads on ESXi hosts with NVIDIA GPUs. In addition, running AI workloads based on NVIDIA GPU Cloud (NGC) containers is specifically validated by VMware.

VMware Private AI Foundation with NVIDIA supports two use cases:

Development use case

Cloud administrators and DevOps engineers can provision AI workloads, including Retrieval-Augmented Generation (RAG), in the form of deep learning virtual machines. Data scientists can use these deep learning virtual machines for AI development. See About Deep Learning VM Images in VMware Private AI Foundation with NVIDIA.

Production use case

Cloud administrators can provide DevOps engineers with a VMware Private AI Foundation with NVIDIA environment for provisioning production-ready AI workloads on Tanzu Kubernetes Grid (TKG) clusters on vSphere IaaS control plane.

For information about the components that are part of the VMware Private AI Foundation with NVIDIA solution and their architecture on top of VMware Cloud Foundation, see System Architecture of VMware Private AI Foundation with NVIDIA.

Preparing VMware Cloud Foundation for Private AI Workload Deployment

2

As a cloud administrator, you must deploy specific software and configure the target VI workload domains so that data scientists and DevOps engineers can deploy AI workloads on top of VMware Private AI Foundation with NVIDIA.

VMware Components in VMware Private AI Foundation with NVIDIA

The functionality of the VMware Private AI Foundation with NVIDIA solution is available across several software components.

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

For information about the VMware Private AI Foundation with NVIDIA architecture and components, see System Architecture of VMware Private AI Foundation with NVIDIA.

Deployment Workflows for VMware Private AI Foundation with NVIDIA

The functionality of VMware Private AI Foundation with NVIDIA is based on a foundational set of components with additional components required to enable the deployment of one of the following AI workload type:

- Deep learning VMs in general
- AI workloads on a GPU-accelerated TKG cluster in general
- RAG workloads as deep learning VMs or applications on GPU-accelerated TKG clusters

  The deployment of a RAG workload extends the general approach for deep learning VMs and AI workloads on TKG clusters with the deployment of a pgvector PostgreSQL database and configuring the application with the pgvector database.

In a disconnected environment, you must take additional steps to set up and deploy appliances and provide resources locally, so that your workloads can access them.

Connected Environment

Task	AI Workload Deployment Use Cases	Steps
Review the architecture and requirements for deploying VMware Private AI Foundation with NVIDIA.	All	■ System Architecture of VMware Private AI Foundation with NVIDIA ■ Requirements for Deploying VMware Private AI Foundation with NVIDIA
Configure a License Service instance on the NVIDIA Licensing Portal and generate a client configuration token.		NVIDIA License System User Guide.
Generate an API key for access to the NVIDIA NGC catalog.		Pulling and Running NVIDIA AI Enterprise Containers
Create a content library for deep learning VM images.	Deploy a deep learning VM	Create a Content Library with Deep Learning VM Images for VMware Private AI Foundation with NVIDIA
Enable vSphere IaaS control plane (formely known as vSphere with Tanzu).	All	Configure vSphere IaaS Control Plane for VMware Private AI Foundation with NVIDIA
Deploy Deploy VMware Aria Automation by using VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.	All Required if data scientists and DevOps engineers will deploy workloads by using self-service catalog items in VMware Aria Automation.	1 Private Cloud Automation for VMware Cloud Foundation 2 Set Up VMware Aria Automation for VMware Private AI Foundation with NVIDIA
Deploy VMware Aria Operations by using VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.	All	Intelligent Operations Management for VMware Cloud Foundation.
Deploy VMware Data Services Manager	Deploy a RAG workload	1 Installing and Configuring VMware Data Services Manager You deploy a VMware Data Services Manager instance in the management domain. 2 Create a Vector Database Catalog Item in VMware Aria Automation
Set up a machine that has access to the Supervisor instance, and has Docker, Helm, and Kubernetes CLI Tools for vSphere.	All Required if the AI workloads will be deployed by directly using the `kubectl` command.	Install the Kubernetes CLI Tools for vSphere

Disconnected Environment

Task	Related AI Workload Deployment Options	Steps
Review the requirements for deploying VMware Private AI Foundation with NVIDIA.	All	■ System Architecture of VMware Private AI Foundation with NVIDIA ■ Requirements for Deploying VMware Private AI Foundation with NVIDIA
Deploy an NVIDIA Delegated License Service Instance.		Installing and Configuring the DLS Virtual Appliance You can deploy the virtual appliance in the same workload domain as the AI workloads or in the management domain.
1 Register an NVIDIA DLS instance on the NVIDIA Licensing Portal, and bind and install a license server on it. 2 Generate a client configuration token.		■ Configuring a Service Instance ■ Managing Licenses on a License Server.
Create a content library for deep learning VM images	Deploy a deep learning VM	Create a Content Library with Deep Learning VM Images for VMware Private AI Foundation with NVIDIA
Enable vSphere IaaS control plane (formely known as vSphere with Tanzu)	All	Configure vSphere IaaS Control Plane for VMware Private AI Foundation with NVIDIA
■ Set up a machine that has access to the Internet and has Docker and Helm installed. ■ Set up a machine that has access to vCenter Server for the VI workload domain, the Supervisor instance, and the local container registry. The machine must have Docker, Helm, and Kubernetes CLI Tools for vSphere.		■ Deploying a Bastion Host ■ Install the Kubernetes CLI Tools for vSphere
Configure a content library for Tanzu Kubernetes releases (TKr) for Ubuntu	■ Deploy a RAG workload on a GPU-accelerated TKG cluster ■ Deploy AI workloads on a GPU-accelerated TKG cluster	Configure a Content Library with Ubuntu TKr for a Disconnected VMware Private AI Foundation with NVIDIA Environment

<table>
<tr><th>Task</th><th>Related AI Workload Deployment Options</th><th>Steps</th></tr>
<tr><td>Set up a Harbor registry service in the Supervisor.</td><td>All
Required if the AI workloads will be deployed on a Supervisor in vSphere IaaS control plane
In an environment without vSphere IaaS control plane, for pulling container images on a deep learning VM running directly on a vSphere cluster, you must configure a registry from another vendor.</td><td>Setting Up a Private Harbor Registry in VMware Private AI Foundation with NVIDIA</td></tr>
<tr><td>Upload the components of the NVIDIA operators to the environment.</td><td>Deploy a RAG workload on a GPU-accelerated TKG cluster
Deploy AI workloads on a GPU-accelerated TKG cluster</td><td>Upload the NVIDIA GPU Operator Components to a Disconnected Environment</td></tr>
<tr><td>Provide a location to download the vGPU guest drivers from.</td><td>Deploy a deep learning VM</td><td>Upload to a local Web server the required vGPU guest driver versions, downloaded from the NVIDIA Licensing Portal, and an index in one of the following formats:
An index <code>.txt</code> file with a list of the <code>.run</code> or <code>.zip</code> files of the vGPU guest drivers.<pre>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</pre>
A directory index in the format generated by Web servers, such as NGINX and Apache HTTP Server. The version-specific vGPU driver files must provided as <code>.zip</code> files.</td></tr>
<tr><td>Upload the NVIDIA NGC container images to a private container registry, such as the Harbor Registry service of the Supervisor.</td><td>All
In an environment without vSphere IaaS control plane, for pulling container images on a deep learning VM running directly on a vSphere cluster, you must configure a registry from another vendor.</td><td>Upload AI Container Images to a Private Harbor Registry in VMware Private AI Foundation with NVIDIA</td></tr>
<tr><td>Deploy VMware Aria Automation by using VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.</td><td>All
Required if data scientists and DevOps engineers will deploy workloads by using self-service catalog items in VMware Aria Automation.</td><td>Private Cloud Automation for VMware Cloud Foundation
Set Up VMware Aria Automation for VMware Private AI Foundation with NVIDIA</td></tr>
</table>

Task	Related AI Workload Deployment Options	Steps
Deploy VMware Aria Operations by using VMware Aria Suite Lifecycle in VMware Cloud Foundation mode.	All	Intelligent Operations Management for VMware Cloud Foundation
Deploy VMware Data Services Manager	Deploy a RAG workload	1 Installing and Configuring VMware Data Services Manager You deploy a VMware Data Services Manager instance in the management domain. 2 Create a Vector Database Catalog Item in VMware Aria Automation

Read the following topics next:

- System Architecture of VMware Private AI Foundation with NVIDIA
- Requirements for Deploying VMware Private AI Foundation with NVIDIA
- Create a Content Library with Deep Learning VM Images for VMware Private AI Foundation with NVIDIA
- Configure vSphere IaaS Control Plane for VMware Private AI Foundation with NVIDIA
- Configure a Content Library with Ubuntu TKr for a Disconnected VMware Private AI Foundation with NVIDIA Environment
- Setting Up a Private Harbor Registry in VMware Private AI Foundation with NVIDIA
- Upload the NVIDIA GPU Operator Components to a Disconnected Environment
- Set Up VMware Aria Automation for VMware Private AI Foundation with NVIDIA

System Architecture of VMware Private AI Foundation with NVIDIA

VMware Private AI Foundation with NVIDIA runs on top of VMware Cloud Foundation adding support for AI workloads in VI workload domains with vSphere IaaS control plane provisioned by using kubectl and VMware Aria Automation .

Figure 2-1. Example Architecture for VMware Private AI Foundation with NVIDIA

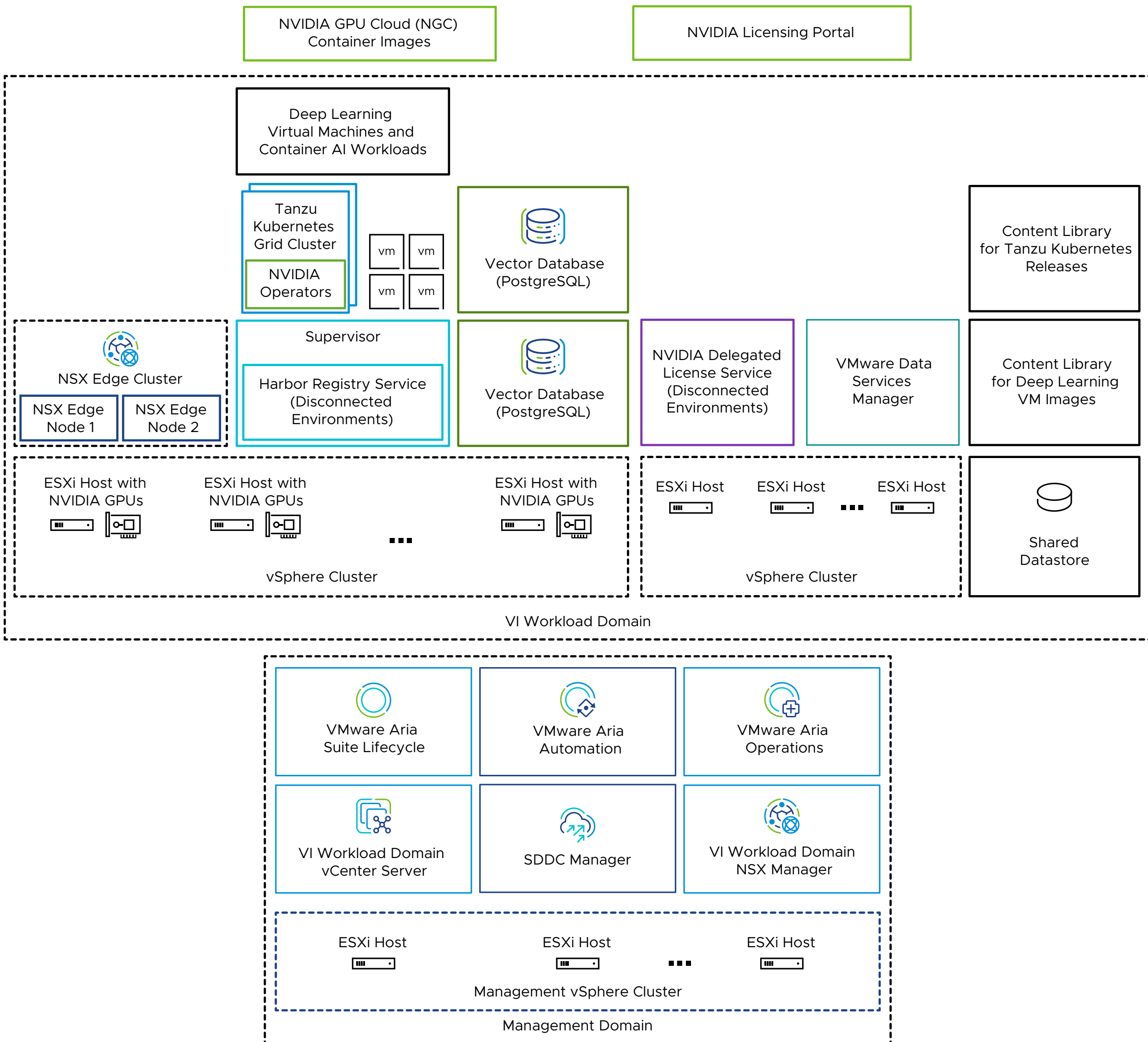


Table 2-1. Components for Running AI Workloads in VMware Private AI Foundation with NVIDIA

Component	Description
GPU-enabled ESXi hosts	ESXi hosts that configured in the following way: ■ Have an NVIDIA GPU that is supported for VMware Private AI Foundation with NVIDIA. The GPU is shared between workloads by using the time slicing or Multi-Instance GPU (MIG) mechanism. ■ Have the NVIDIA vGPU host manager driver installed so that you can use vGPU profiles based on MIG or time slicing.
Supervisor	One or more vSphere clusters enabled for vSphere IaaS control plane so that you can run virtual machines and containers on vSphere by using the Kubernetes API. A Supervisor is a Kubernetes cluster itself, serving as the control plane to manage workload clusters and virtual machines.
Harbor registry	A local image registry in a disconnected environment where you host the container images downloaded from the NVIDIA NGC catalog.
NSX Edge cluster	A cluster of NSX Edge nodes that provides 2-tier north-south routing for the Supervisor and the workloads it runs. The Tier-0 gateway on the NSX Edge cluster is in active-active mode.
NVIDIA Operators	■ NVIDIA GPU Operator. Automates the management of all NVIDIA software components needed to provision GPU to containers in a Kubernetes cluster. NVIDIA GPU Operator is deployed on a TKG cluster. ■ NVIDIA Network Operator. NVIDIA Network Operator also helps configuring the right mellanox drivers for containers using virtual functions for high speed networking, RDMA and GPUDirect. Network Operator works together with the GPU Operator to enable GPUDirect RDMA on compatible systems. NVIDIA Network Operator is deployed on a TKG cluster.
Vector database	A PostgreSQL database that has the pgvector extension enabled so that you can use it in Retrieval Augmented Generation (RAG) AI workloads.

Table 2-1. Components for Running AI Workloads in VMware Private AI Foundation with NVIDIA (continued)

Component	Description
■ NVIDIA Licensing Portal ■ NVIDIA Delegated License Service (DLS)	You use the NVIDIA Licensing Portal to generate a client configuration token to assign a license to the guest vGPU driver in the deep learning virtual machine and the GPU Operators on TKG clusters. In a disconnected environment or to have your workloads getting license information without using an Internet connection, you host the NVIDIA licenses locally on a Delegated License Service (DLS) appliance.
Content library	Content libraries store the images for the deep learning virtual machines and for the Tanzu Kubernetes releases. You use these images for AI workload deployment within the VMware Private AI Foundation with NVIDIA environment. In a connected environment, content libraries pull their content from VMware managed public content libraries. In a disconnected environment, you must upload the required images manually or pull them from an internal content library mirror server.
NVIDIA GPU Cloud (NGC) catalog	A portal for GPU-optimized containers for AI, and machine learning that are tested and ready to run on supported NVIDIA GPUs on premises on top of VMware Private AI Foundation with NVIDIA.

As a cloud administrator, you use the management components in VMware Cloud Foundation

Table 2-2. Management Components in VMware Private AI Foundation with NVIDIA

Management Component	Description
SDDC Manager	You use SDDC Manager for the following tasks: ■ Deploy a GPU-enabled VI workload domain that is based vSphere Lifecycle Manager images and add clusters to it. ■ Deploy an NSX Edge cluster in VI workload domains for use by Supervisor instances and in the management domain for the VMware Aria Suite components of VMware Private AI Foundation with NVIDIA. ■ Deploy a VMware Aria Suite Lifecycle instance which is integrated with the SDDC Manager repository.
VI Workload Domain vCenter Server	You use this vCenter Server instance to enable and configure a Supervisor.
VI Workload Domain NSX Manager	SDDC Manager uses this NSX Manager to deploy and update NSX Edge clusters.

Table 2-2. Management Components in VMware Private AI Foundation with NVIDIA (continued)

Management Component	Description
VMware Aria Suite Lifecycle	You use VMware Aria Suite Lifecycle to deploy and update VMware Aria Automation and VMware Aria Operations.
VMware Aria Automation	You use VMware Aria Automation to add self-service catalog items for deploying AI workloads for DevOps engineers and data scientists.
VMware Aria Operations	You use VMware Aria Operations for monitoring the GPU consumption in the GPU-enabled workload domains.
VMware Data Services Manager	You use VMware Data Services Manager to create vector databases, such as a PostgreSQL database with pgvector extension.

Requirements for Deploying VMware Private AI Foundation with NVIDIA

You deploy components of VMware Private AI Foundation with NVIDIA in your VMware Cloud Foundation environment in a VI workload domain where you must have certain NVIDIA components installed.

Required VMware Software Versions

See VMware Components in VMware Private AI Foundation with NVIDIA.

Supported NVIDIA GPU Devices

Before you start using VMware Private AI Foundation with NVIDIA, make sure that the GPUs on your ESXi hosts are supported by VMware by Broadcom:

Table 2-3. Supported NVIDIA Components for VMware Private AI Foundation with NVIDIA

NVIDIA Component	Supported Options
NVIDIA GPUs	■ NVIDIA A100 ■ NVIDIA L40S ■ NVIDIA H100
GPU sharing mode	■ Time slicing ■ Multi-Instance GPU (MIG)

Required NVIDIA Software

The GPU device must support the latest NVIDIA AI Enterprise (NVAIE) vGPU profiles. See the NVIDIA Virtual GPU Software Supported GPUs documentation for guidance.

- NVIDIA vGPU host driver (including the VIB for ESXi hosts), that is compatible with your VMware Cloud Foundation version. See Virtual GPU Software for VMware vSphere Release Notes.
- NVIDIA GPU Operator that is compatible with the Kubernetes version of the deployed TKG clusters. See NVIDIA GPU Operator Release Notes and VMware Tanzu Kuberenetes releases Release Notes.

Required VMware Cloud Foundation Setup

Before you deploy VMware Private AI Foundation with NVIDIA, a specific configuration must be available in VMware Cloud Foundation.

- A VMware Cloud Foundation license.
- A VMware Private AI Foundation with NVIDIA add-on license.

  You need the VMware Private AI Foundation with NVIDIA add-on license to access the following functionality:

  - Private AI setup in VMware Aria Automation for catalog items for easy provisioning of GPU-accelerated deep learning virtual machines and TKG clusters.
  - Provisioning of PostgreSQL databases with the pgvector extension with enterprise support.
  - Deploying and using the deep learning virtual machine image delivered by VMware by Broadcom.

  You can deploy AI workloads with and without a Supervisor enabled and use the GPU metrics in vCenter Server and VMware Aria Operations under the VMware Cloud Foundation license.
- Licensed NVIDIA vGPU product including the host driver VIB file for ESXi hosts and the guest OS drivers. See the NVIDIA Virtual GPU Software Supported GPUs documentation for guidance.
- The VIB file of the NVIDIA vGPU host driver downloaded from https://nvid.nvidia.com/
- A vSphere Lifecycle Manager image with the VIB file of the vGPU host manager driver available in SDDC Manager. See Managing vSphere Lifecycle Manager Images in VMware Cloud Foundation.
- A VI workload domain with at least 3 ESXi GPU-enabled hosts that is based on the vSphere Lifecycle Manager image containing the host manager driver VIB file. See Deploy a VI Workload Domain Using the SDDC Manager UI and Managing vSphere Lifecycle Manager Images in VMware Cloud Foundation.

- NVIDIA vGPU host driver installed and vGPU configured on each ESXi host in the cluster for AI workloads.

  a On each ESXi host, enable SR-IOV in the BIOS and Shared Direct on the graphics devices for AI operations.

    For information about configuring SR-IOV, see the documentation from your hardware vendor. For information about configuring Shared Direct on graphics devices, see Configure Virtual Graphics on vSphere.

  b Install the NVIDIA vGPU host manager driver on each ESXi host in one of the following ways:

    - Install the driver on each host and add the VIB file of the driver to the vSphere Lifecycle image for the cluster.

      See NVIDIA Virtual GPU Software Quick Start Guide.

    - Add the VIB file of the driver to the vSphere Lifecycle image for the cluster and remediate the hosts.

  c If you want to use the Multi-Instance GPU (MIG) sharing, enable it on each ESXi host in the cluster.

    See NVIDIA MIG User Guide.

  d On the vCenter Server instance for VI workload domain, set the `vgpu.hotmigrate.enabled` advanced setting to `true` so that virtual machines with vGPU can be migrated by using vSphere vMotion.

    See Configure Advanced Settings.

Create a Content Library with Deep Learning VM Images for VMware Private AI Foundation with NVIDIA

Deep learning VM images in VMware Private AI Foundation with NVIDIA are distributed in a shared content library published by VMware. As a cloud administrator, you use a content library to pull specific VM images in your VI workload domain during VM deployment.

Prerequisites

As a cloud administrator, verify that VMware Private AI Foundation with NVIDIA is deployed and configured. See Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment.

Procedure

1 Log in to the vCenter Server instance for the VI workload domain at **https://<vcenter_server_fqdn>/ui**.

2 Select **Menu > Content Libraries** and click **Create**.

3 Create a content library for the deep learning VM images.

- For a connected environment, create a subscribed content library that is connected to `https://packages.vmware.com/dl-vm/lib.json`. Authentication is not required.
- For a disconnected environment, download the deep learning VM images from `https://packages.vmware.com/dl-vm/` and import them in to a local content library.

  For each image, download the relevant `.ovf`, `.vmdk`, `.mf`, and `.cert` files.

Configure vSphere IaaS Control Plane for VMware Private AI Foundation with NVIDIA

To provide DevOps engineers and data scientists with the ability to deploy deep learning virtual machines or TKG clusters with AI container workloads, you must deploy a Supervisor on a GPU-enabled cluster in a VI workload domain and create vGPU-enabled VM classes.

Prerequisites

See Requirements for Deploying VMware Private AI Foundation with NVIDIA.

Procedure

1 Deploy an NSX Edge Cluster in the VI workload domain by using SDDC Manager.

SDDC Manager also deploys a Tier-0 gateway that you specify at Supervisor deployment. The Tier-0 gateway is in active-active high availability mode.

2 Configure a storage policy for the Supervisor.

See Create Storage Policies for vSphere with Tanzu.

3 Deploy a Supervisor on a cluster of GPU-enabled ESXi hosts in the VI workload domain.

You use static IP address assignment for the management network. Assign the supervisor VM management network on the vSphere Distributed Switch for the cluster.

Configure the workload network in the following way:

- Use the vSphere Distributed Switch for the cluster or create one specifically for AI workloads.
- Configure the Supervisor with the NSX Edge cluster and Tier-0 gateway that you deployed by using SDDC Manager.
- Set the rest of the values according to your design.

Use the storage policy you created.

For more information on deploying a supervisor on a single cluster, see Deploy a One-Zone Supervisor with NSX Networking.

4 Configure vGPU-based VM classes for AI workloads.

In these VM classes, you set the compute requirements and a vGPU profile for an NVIDIA GRID vGPU device according to the vGPU devices configured on the ESXi hosts in the Supervisor cluster.

- For information about setting up vGPU-based VM classes for virtual machines, see Create a Custom VM Class Using the vSphere Client and Add PCI Devices to a VM Class in vSphere with Tanzu.
- For information about setting up vGPU-based VM classes for TKG worker nodes, see Create a Custom VM Class with a vGPU Profile in vSphere 8 Update 2b and later and Configuring vSphere Namespaces for TKG Clusters on Supervisor.

For the VM class for deploying deep learning VMs with NVIDIA RAG workloads, set the following additional settings in the VM class dialog box:

- Select the full-sized vGPU profile for time-slicing mode or a MIG profile. For example, for NVIDIA A100 40GB card in vGPU time-slicing mode, select **nvidia_a100-40c**.
- On the **Virtual Hardware** tab, allocate more than 16 virtual CPU cores and 64 GB of virtual memory.
- On the **Advanced Parameters** tab, set the `pciPassthru<vgpu-id>.cfg.enable_uvm` parameter to **1**.

  where `<vgpu-id>` identifies the vGPU assigned to the virtual machine. For example, if two vGPUs are assigned to the virtual machine, you set `pciPassthru0.cfg.parameter=1` and `pciPassthru1.cfg.parameter = 1`.

5 If you plan to use the `kubectl` command line tool to deploy a deep learning VM or an GPU-accelerated TKG cluster on a Supervisor, create and configure a vSphere namespace, adding resource limits, storage policy, permissions for DevOps engineers, and associating the vGPU-based VM classes with it.

- For information about setting up vSphere namespaces for virtual machines, see Create and Configure a vSphere Namespace on the Supervisor.
- For information about setting up vSphere namespaces for TKG clusters, see Configuring vSphere Namespaces for TKG Clusters on Supervisor.

6 If you plan to enable deployments of deep learning VMs on a Supervisor by directly calling `kubectl`, add the content library to the vSphere namespace for AI workloads.

VMware Aria Automation creates a namespace every time a deep learning VM is provisioned, automatically adding the content library to it.

a Select **Menu > Workload Management**.

b Navigate to the namespace for AI workloads.

c On the **VM Service** card, click **Manage content libraries**.

d Select the content library with the deep learning VM images and click **OK**.

Configure a Content Library with Ubuntu TKr for a Disconnected VMware Private AI Foundation with NVIDIA Environment

As a cloud administrator, if your environment has no Internet connectivity, you provide a local content library where you manually upload Tanzu Kubernetes releases (TKr) and associate it with the Supervisor.

Deploying NVIDIA-aware AI workloads on TKG clusters requires the use of the Ubuntu edition of Tanzu Kubernetes releases.

Caution The TKr content library is used across all vSphere namespaces in the Supervisor when you provision new TKG clusters.

Prerequisites

As a cloud administrator, verify that VMware Private AI Foundation with NVIDIA is deployed and configured. See Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment

Procedure

1. Download the Ubuntu-based TKr images with the required Kubernetes versions from `https://wp-content.vmware.com/v2/latest/`.
2. Log in to the vCenter Server instance for the VI workload domain at **http://<vcenter_server_fqdn>/ui**.
3. Create a local content library and import the TKr images there.

   See Create a Local Content Library (for Air-Gapped Cluster Provisioning).
4. Add the content library to the Supervisor.
   1. Select **Menu > Workload Management**.
   2. Navigate to the Supervisor for AI workloads.
   3. On the **Configure** tab, select **General**.
   4. Next to the **Tanzu Kubernetes Grid Service** property, click **Edit**.
   5. On the **General** page that appears, expand **Tanzu Kubernetes Grid Service**, and next to **Content Library**, click **Edit**.
   6. Select the content library with the TKr images and click **OK**.

Setting Up a Private Harbor Registry in VMware Private AI Foundation with NVIDIA

You can use Harbor as a Supervisor Service as a local registry for container images from the NVIDIA NGC catalog.

Note The installation of the Harbor service in the Supervisor requires an Internet connection.

If you want to use the Harbor registry integration with Supervisor, you can follow these setup approaches:

- Use a Harbor registry only in the Supervisor in the GPU-enabled workload domain. Perform the following tasks:

  a Enable Harbor as a Supervisor Service.

  b Upload AI Container Images to a Private Harbor Registry in VMware Private AI Foundation with NVIDIA

  You can disconnect your environment from the Internet and start using the Harbor service as a local container registry after you install the service or after you install it and download the initial set of required container images.

  In this approach, you must manually download container images from the NVIDIA NGC catalog to a machine in the environment and then upload them to the registry.

- Create a Harbor Registry in VMware Private AI Foundation with NVIDIA as a Replica of a Connected Registry.

  One Harbor registry, running outside the VMware Private AI Foundation with NVIDIA environment, is always connected to the Internet. The Harbor registry in the Supervisor for the GPU-enabled workload domain receives container images from the connected one using a proxy mechanism. In this way, the main components of the VMware Cloud Foundation instance remain isolated.

  In this approach, additional resources are required for the connected registry.

Note Allocate enough storage space for hosting the NVIDIA NGC containers you plan to deploy on a deep learning VM or on a TKG cluster. Accommodate at least three versions of each container in the storage space.

If connecting to the Internet while installing the Harbor service or setting up a connected Harbor registry is not an option for your organization, use a container registry by another vendor.

Upload AI Container Images to a Private Harbor Registry in VMware Private AI Foundation with NVIDIA

In a disconnected environment where you use a Harbor registry only on the AI-ready Supervisor, you must manually upload the AI container images that you intend to deploy on a deep learning VM or a TKG cluster from the NVIDIA NGC catalog to Harbor.

Procedure

1 On the machines for access to NVIDIA NGC and to the disconnected VMware Cloud Foundation instance, configure the Docker client with the certificate of the Harbor registry.

   See Configure a Docker Client with a Registry Certificate.

2 Log in to NVIDA NGC.

   Use the reserved user name of $oauthtoken and paste the API key in the password field.

```
docker login nvcr.io
```

3 Pull the required container images to the machine with access to NVIDIA NGC catalog and save them to an archive.

   For example, to download the CUDA Sample container image, run the following commands.

```
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 Copy the archive to the machine with access to the local container registry.

5 On the machine with access to the local container registry, load the container image.

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

6 Log in to the Harbor registry.

   For example, if the Harbor registry is running at my-harbor-registry.example.com, run the following commands.

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

7 Tag the image that you want to push to the project with the same name as the namespace where you want to use it.

   For example, to tag the CUDA Sample container image as latest for the my-private-ai-namespace project on the my-harbor-registry.example.com registry, run the following command.

```
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 Push the container images to the Harbor registry.

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

Create a Harbor Registry in VMware Private AI Foundation with NVIDIA as a Replica of a Connected Registry

To be able to update easily to the latest images in the NVIDIA NGC catalog, you can use a Harbor registry in a Supervisor that is in another VI workload domain or VMware Cloud Foundation instance and can be connected to Internet. You then replicate this connected registry on the Supervisor where you plan to run AI workloads.

You pull the latest container images from NVIDIA NGC to the connected Harbor registry and transfer them to the disconnected one by using a proxy-cached connection. In this way, you do not have to download container images and then upload them manually on a frequent basis.

Note You can also use a connected container registry by another vendor.

You set up the network between the two registries in the following way:

- The connected registry is routable to the replica registry.
- The connected registry is placed in a DMZ where only `docker push` and `docker pull` communication is allowed between the two registries.

Prerequisites

Enable Harbor as a Supervisor Service in the Supervisor in the GPU-enabled workload domain.

Procedure

1. Log in to the connected Harbor Registry UI as a Harbor system administrator.
2. Go to the **Administration** > **Registries** page and create an endpoint for the NVIDIA NGC catalog `nvcr.io/nvaie` selecting the **Docker Registry** provider and with your NVIDIA NGC API key.
3. Go to the **Administration** > **Projects** page and create a proxy-cache project, connected to the endpoint for `nvcr.io/nvaie`.
4. Back on the **Registries** page, create a replication endpoint for the disconnected registry, selecting the **Harbor** provider.
5. Go to the **Administration** > **Replications** page and create a replication rule.
   - Use push-based replication mode.
   - In the **Destination registry** property, enter the URL of the disconnected registry on the AI-ready Supervisor.
   - Set filters, target namespace and trigger mode according to the requirements of your organization.

What to do next

1. Pull the container images that are required by your organization from NVIDIA NGC to the connected registry by running `docker pull` on the Docker client machine.
2. If the replication rule has manual trigger mode, run manually replications as needed.

Upload the NVIDIA GPU Operator Components to a Disconnected Environment

In a disconnected environment, upload the components of the NVIDIA GPU Operator on internal locations.

Procedure

1 Provide a local Ubuntu package repository and upload the container images in the NVIDIA GPU Operator package to the Harbor Registry for the Supervisor.

2 Provide a local Helm chart repository with NVIDIA GPU Operator chart definitions.

3 Update the Helm chart definitions of the NVIDIA GPU Operator to use the local Ubuntu package repository and private Harbor Registry.

Results

For more information, see Installing VMware vSphere with VMware Tanzu (Air-gapped).

Set Up VMware Aria Automation for VMware Private AI Foundation with NVIDIA

VMware Aria Automation provides support for self-service catalog items that DevOps engineers and data scientists can use to provision AI workloads in VMware Private AI Foundation with NVIDIA in a user-friendly and customizable way.

Prerequisites

As a cloud administrator, verify that the VMware Private AI Foundation with NVIDIA environment is configured. See Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment.

Procedure

1 Connect VMware Aria Automation to a Workload Domain for VMware Private AI Foundation with NVIDIA

   Before you can add the catalog items for provisioning AI applications by using VMware Aria Automation, you connect VMware Aria Automation to VMware Cloud Foundation.

2 Create AI Self-Service Catalog Items in VMware Aria Automation

   As a cloud administrator, you use the catalog setup wizard for private AI in VMware Aria Automation to quickly add catalog items for deploying deep learning virtual machines or GPU-accelerated TKG clusters in a VI workload domain in the connected VMware Cloud Foundation.

3 Create a Vector Database Catalog Item in VMware Aria Automation

   As a cloud administrator, you can add a catalog item for provisioning databases in VMware Data Services Manager to the Automation Service Broker in VMware Aria Automation.

Connect VMware Aria Automation to a Workload Domain for VMware Private AI Foundation with NVIDIA

Before you can add the catalog items for provisioning AI applications by using VMware Aria Automation, you connect VMware Aria Automation to VMware Cloud Foundation.

Procedure

- In the VMware Aria Automation UI, run quick start wizard for VMware Cloud Foundation or for VMware vCenter Server .

  See How do you get started with VMware Aria Automation using the VMware Cloud Foundation Quickstart or How do you get started with VMware Aria Automation using the VMware vCenter Server Quickstart in the *Getting Started with VMware Aria Automation* documentation.

Create AI Self-Service Catalog Items in VMware Aria Automation

As a cloud administrator, you use the catalog setup wizard for private AI in VMware Aria Automation to quickly add catalog items for deploying deep learning virtual machines or GPU-accelerated TKG clusters in a VI workload domain in the connected VMware Cloud Foundation.

Data scientists can use deep learning catalog items for deployment of deep learning VMs. DevOps engineers can use the catalog items for provisioning AI-ready TKG clusters.

Every time you run it, the catalog setup wizard for private AI adds items for deep learning virtual machines and TKG clusters to the Service Broker catalog. You can run the wizard every time you need the following:

- Enable provisioning of AI workloads on another supervisor.
- Accommodate a change in your NVIDIA AI Enterprise license, including the client configuration `.tok` file and license server, or the download URL for the vGPU guest drivers for a disconnected environment.
- Accommodate a deep learning VM image change.
- Use other vGPU or non-GPU VM classes, storage policy, or container registry.
- Create catalog items in a new project.

Note VMware Aria Automation creates a vSphere namespace every time a deep learning VM or a Tanzu Kubernetes Grid cluster is provisioned.

Procedure

- Add Private AI items to the Automation Service Broker catalog.

What to do next

By using the Automation Service Broker, your data scientists can proceed with deploying deep learning VMs and your DevOps engineers - with provisioning GPU-enabled Tanzu Kubernetes Grid clusters. See Deploy a non-RAG deep learning virtual machine in VMware Aria Automation.

Create a Vector Database Catalog Item in VMware Aria Automation

As a cloud administrator, you can add a catalog item for provisioning databases in VMware Data Services Manager to the Automation Service Broker in VMware Aria Automation.

Prerequisites

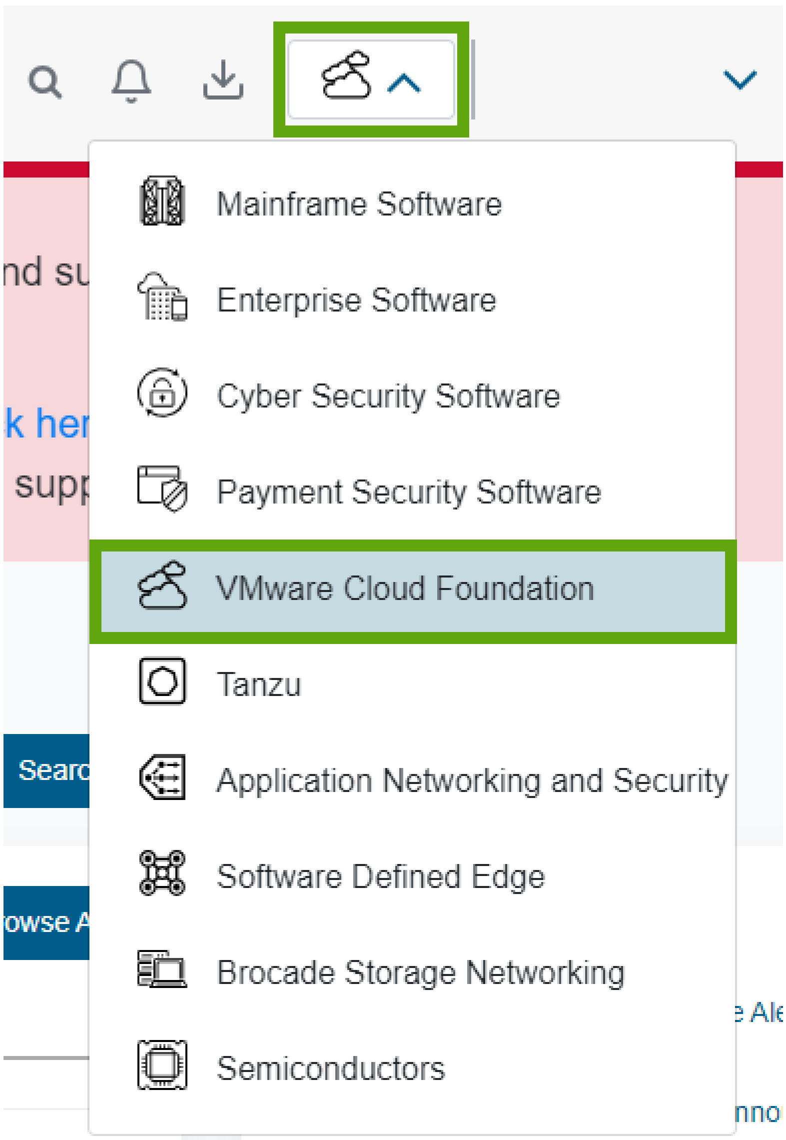
- Verify that you have VMware Data Services Manager 2.1 deployed.
- Provide a machine that has Python 3.10 installed and has access to the VMware Data Services Manager and VMware Aria Automation instances.

Procedure

1 Download the `AriaAutomation_DataServicesManager` bundle for VMware Data Services Manager 2.1 from the Broadcom Technical Portal.

   a Log in to the the Broadcom Support Portal.

   b From the software category drop-down menu in the top right corner of the portal, select **VMware Cloud Foundation**.



   c In the left navigation pane, click **My Downloads**.

   d On the **My Downloads - VMware Cloud Foundation** page, click **VMware Data Services Manager**.

   e Click the release number and download the `AriaAutomation_DataServicesManager` bundle.

2 On the machine running Python, upload the `AriaAutomation_DataServicesManager` bundle and extract its content.

3 Update the `config.json` file in the folder where you extracted the bundle with the URLs and user credentials for VMware Data Services Manager and VMware Aria Automation.

   Optionally, you can also set the name of the catalog item, Automaton Assembler project, and other parameters.

4 To create the catalog items in VMware Aria Automation, run the `aria.py` Python script in the following way.

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

Results

The Python script creates items in VMware Aria Automation that are required for using VMware Data Services Manager for database provisioning. See the `readme.md` file in the `AriaAutomation_DataServicesManager` bundle

What to do next

Your data scientists or DevOps engineers can deploy a vector database from the Automation Service Broker catalog with pgvector extension and integrate it their RAG workloads. See Chapter 5 Deploying RAG Workloads in VMware Private AI Foundation with NVIDIA.

Deploying a Deep Learning VM in VMware Private AI Foundation with NVIDIA

3

VMware Private AI Foundation with NVIDIA supports provisioning pre-configured deep learning VMs that data scientists can use for AI development.

As a data scientist, you have the following options to start using a deep learning VM:

- Deploy a deep learning VM by using a self-service catalog item in VMware Aria Automation.
- Request your DevOps engineer to deploy a deep learning VM on a Tanzu Kubernetes Grid cluster by using the `kubectl` command.
- Request your cloud administrator to deploy a deep learning VM on a vSphere cluster to quickly explore the deep learning VM templates.

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

  The deep learning virtual machine images delivered as part of VMware Private AI Foundation with NVIDIA are preconfigured with popular ML libraries, frameworks, and toolkits, and are optimized and validated by NVIDIA and VMware for GPU acceleration in a VMware Cloud Foundation environment.

- Deploy a Deep Learning VM by Using a Self-Service Catalog in VMware Aria Automation

  In VMware Private AI Foundation with NVIDIA, as a data scientist or DevOps engineer, you can deploy a deep learning VM from VMware Aria Automation by using an AI workstation self-service catalog items in Automation Service Broker.

- Deploy a Deep Learning VM Directly on a vSphere Cluster in VMware Private AI Foundation with NVIDIA

  To quickly give data scientists the opportunity to test the deep learning VM templates in VMware Private AI Foundation with NVIDIA, as a cloud administrator, you can deploy a deep learning VM directly on a vSphere cluster by using the vSphere Client.

- Deploy a Deep Learning VM by Using the `kubectl` Command in VMware Private AI Foundation with NVIDIA

  The VM service in the Supervisor in vSphere IaaS Control Plane enables DevOps engineers to deploy and run deep learning VMs by using the Kubernetes API.

- Customizing Deep Learning VM Deployment in VMware Private AI Foundation with NVIDIA

  When you deploy a deep learning VM in vSphere IaaS control plane by using `kubectl` or directly on a vSphere cluster, you must fill in custom VM properties.

- Troubleshooting Deep Learning VM Deployment in VMware Private AI Foundation with NVIDIA

  The troubleshooting information about deployment of deep learning VM in VMware Private AI Foundation with NVIDIA provides solutions to potential problems that you might encounter.

About Deep Learning VM Images in VMware Private AI Foundation with NVIDIA

The deep learning virtual machine images delivered as part of VMware Private AI Foundation with NVIDIA are preconfigured with popular ML libraries, frameworks, and toolkits, and are optimized and validated by NVIDIA and VMware for GPU acceleration in a VMware Cloud Foundation environment.

As a data scientist, you can use the deep learning VMs provisioned from these images for AI prototyping, fine tuning, validation, and inference.

The software stack for running AI applications on top of NVIDIA GPUs is validated in advance. As a result, you directly start AI developing, without spending time installing and validating the compatibility of operating systems, software libraries, ML frameworks, toolkits, and GPU drivers.

What Does a Deep Learning VM Image Contain?

The latest deep learning virtual machine image contains the following software. For information on the component versions in each deep learning VM image release, see VMware Deep Learning VM Release Notes.

<table>
<tr><th>Software Component Category</th><th colspan="2">Software Component</th></tr>
<tr><td>Embedded</td><td colspan="2">Canonical Ubuntu
NVIDIA Container Toolkit
Docker Community Engine
Miniconda and a PyTorch Conda manifest.</td></tr>
<tr><td rowspan="3">Can be pre-installed automatically when you start the deep learning VM for the first time</td><td colspan="2">vGPU guest driver according to the version of the vGPU host driver</td></tr>
<tr><td rowspan="2">Deep learning (DL) workloads</td><td>CUDA Sample
You can use a deep learning VM with running CUDA samples to explore vector addition, gravitational n-body simulation, or other examples on a VM. See the CUDA Samples page in the NVIDIA NGC catalog.</td></tr>
<tr><td>PyTorch.
You can use a deep learning VM with a PyTorch library to explore conversational AI, NLP, and other types of AI models, on a VM. See the PyTorch page in the NVIDIA NGC catalog.
You can use a ready JupyterLab instance with PyTorch installed and configured at <code>http://dl_vm_ip:8888</code>.</td></tr>
</table>

Software Component Category	Software Component
	TensorFlow. You can use a deep learning VM with a TensorFlow library to explore conversational AI, NLP, and other types of AI models, on a VM. See the TensorFlow page in the NVIDIA NGC catalog. You can use a ready JupyterLab instance with TensorFlow installed and configured at `http://dl_vm_ip:8888`.
	DCGM Exporter You can use a deep learning VM with a Data Center GPU Manager (DCGM) exporter to monitor the health of and get metrics from GPUs used by a DL workload, using NVIDIA DCGM, Prometheus, and Grafana. See the DCGM Exporter page in the NVIDIA NGC catalog. In a deep learning VM, you run the DCGM Exporter container together with a DL workload that performs AI operations. After the deep learning VM is started, DCGM Exporter is ready to collect vGPU metrics and export the data to another application for further monitoring and visualization. For information how to use DGCM Exporter to visualize metrics with Prometheus and Grafana, see DCGM Exporter.
	Triton Inference Server You can use a deep learning VM with a Triton Inference Server for loading a model repository and receive inference requests. See the Triton Inference Server page in the NVIDIA NGC catalog. For information how to use Triton Inference Server for inference requests for AI models, see Triton Inference Server.
	NVIDIA RAG You can use a deep learning VM to build Retrieval Augmented Generation (RAG) solutions with an Llama2 model. See the NVIDIA RAG Applications Docker Compose documentation (requires specific account permissions) . A sample chatbot Web application that you can access at `http://dl_vm_ip:3001/orgs/nvidia/models/text-qa-chatbot`. You can upload your own knowledge base.

Deploying a Deep Learning VM

As a data scientist, you can deploy a deep learning VM on your own by using catalog items in VMware Aria Automation. Otherwise, a cloud administrator or DevOps engineer deploys such a VM for you.

Deploy a Deep Learning VM by Using a Self-Service Catalog in VMware Aria Automation

In VMware Private AI Foundation with NVIDIA, as a data scientist or DevOps engineer, you can deploy a deep learning VM from VMware Aria Automation by using an AI workstation self-service catalog items in Automation Service Broker.

For information about deep learning VM images in VMware Private AI Foundation with NVIDIA, see About Deep Learning VM Images in VMware Private AI Foundation with NVIDIA.

Prerequisites

- Verify that your cloud administrator has set up the VMware Aria Automation catalog for private AI application deployment. See Add Private AI items to the Automation Service Broker catalog.
- Verify the your cloud administrator has assigned the user role that is required for deploying deep learning VMs.

Procedure

- Deploy a non-RAG deep learning virtual machine in VMware Aria Automation or Deploy a Deep Learning VM with a RAG Workload.

  Deploying a deep learning VM with NVIDIA RAG requires a vector database, such as a PostgreSQL database with pgvector in VMware Data Services Manager.

Results

The vGPU guest driver and the specified deep learning workload are installed the first time you start the deep learning VM.

What to do next

For details on how to access the virtual machine and the JupyterLab instance that comes with some of the deep learning VM images, in Automation Service Broker, navigate to **Consume > Deployments > Deployments**.

Deploy a Deep Learning VM Directly on a vSphere Cluster in VMware Private AI Foundation with NVIDIA

To quickly give data scientists the opportunity to test the deep learning VM templates in VMware Private AI Foundation with NVIDIA, as a cloud administrator, you can deploy a deep learning VM directly on a vSphere cluster by using the vSphere Client.

For information about deep learning VM images in VMware Private AI Foundation with NVIDIA, see About Deep Learning VM Images in VMware Private AI Foundation with NVIDIA.

Deploying a deep learning VM with NVIDIA RAG requires a vector database, such as a PostgreSQL database with pgvector in VMware Data Services Manager. For information about deploying such a database and integrating it in a deep learning VM, see Deploy a Deep Learning VM with a RAG Workload.

Prerequisites

Verify that VMware Private AI Foundation with NVIDIA is deployed and configured. See Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment.

Procedure

1 Log in to the vCenter Server instance for the VI workload domain.

2 From the vSphere Client home menu, select **Content Libraries**.

3 Navigate to the deep learning VM image in the content library.

4 Right-click an OVF template and select **New VM from This Template**.

5 On the **Select name and folder** page of wizard that appears, enter a name and select a VM folder, select **Customize this virtual machine's hardware**, and click **Next**.

6 Select a GPU-enabled cluster in the VI workload domain, select if the virtual machine must be powered-on after deployment is complete, and click **Next**.

7 Follow the wizard to select a datastore and a network on the distributed switch for the cluster.

8 On the **Customize template** page, enter the custom VM properties that are required for setting up the AI functionality and click **Next**.

   See OVF Properties of Deep Learning VMs.

9 On the **Customize hardware** page, assign an NVIDIA vGPU device to the virtual machine as a **New PCI Device** and click **Next**.

   For a deep learning VM that is running an NVIDIA RAG, select the full-sized vGPU profile for time-slicing mode or a MIG profile. For example, for NVIDIA A100 40GB in vGPU time-slicing mode, select **nvidia_a100-40c**.

10 For a deep learning VM that is running an NVIDIA RAG, in the **Advanced Parameters** tab of the virtual machine settings, set the `pciPassthru<vgpu-id>.cfg.enable_uvm` parameter to **1**.

   where `<vgpu-id>` identifies the vGPU assigned to the virtual machine. For example, if two vGPUs are assigned to the virtual machine, you set `pciPassthru0.cfg.parameter=1` and `pciPassthru1.cfg.parameter = 1`.

11 Review the deployment specification and click **Finish**.

Results

The vGPU guest driver and the specified deep learning workload is installed the first time you start the deep learning VM.

You can examine the logs or open the JupyterLab instance that comes with some of the images. You can share access details with data scientists in your organization. See Deep Learning Workloads in VMware Private AI Foundation with NVIDIA.

What to do next

- Connect to the deep learning VM over SSH and verify that all components are installed and running as expected.
- Send access details to your data scientists.

Deploy a Deep Learning VM by Using the `kubectl` Command in VMware Private AI Foundation with NVIDIA

The VM service in the Supervisor in vSphere IaaS Control Plane enables DevOps engineers to deploy and run deep learning VMs by using the Kubernetes API.

As a DevOps engineer, you use `kubectl` to deploy a deep learning VM on the namespace configured by the cloud administrator.

For information about deep learning VM images in VMware Private AI Foundation with NVIDIA, see About Deep Learning VM Images in VMware Private AI Foundation with NVIDIA.

Deploying a deep learning VM with NVIDIA RAG requires a vector database, such as a PostgreSQL database with pgvector in VMware Data Services Manager. For information about deploying such a database and integrating it in a deep learning VM, see Deploy a Deep Learning VM with a RAG Workload.

Prerequisites

Verify with the cloud administrator that the VMware Private AI Foundation with NVIDIA is deployed and configured. See Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment.

Procedure

1 Log in to the Supervisor control plane.

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

2 Examine that all required VM resources, such as VM classes and VM images, are in place on the namespace.

See View VM Resources Available on a Namespace in vSphere with Tanzu.

3 Prepare the YAML file for the deep learning VM.

Use the vm-operator-api, setting the OVF properties as a ConfigMap object. For information on available OVF properties, see OVF Properties of Deep Learning VMs.

For example, you can create a YAML specification `example-dl-vm.yaml` for an example deep learning VM running PyTorch in a connected environment.

```
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
```

Note `user-data` is the base64 encoded value for the following cloud-init code:

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

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

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

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

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

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

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

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

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

4 Switch to the context of the vSphere namespace created by the cloud administrator.

   For example, for a namespace called `example-dl-vm-namespace`:

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

5 Deploy the deep learning VM.

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

6 Verify that the VM has been created by running these commands.

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

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

7 Ping the IP address of the virtual machine assigned by the requested networking service.

   To get the public address and the ports for access to the deep learning VM, get the details about the load balancer service that has been created.

```
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
```

Results

The vGPU guest driver and the specified DL workload is installed the first time you start the deep learning VM.

What to do next

- You can examine the logs or open the JupyterLab notebook that comes with some of the images. See Deep Learning Workloads in VMware Private AI Foundation with NVIDIA.
- Send access details to your data scientists.

Customizing Deep Learning VM Deployment in VMware Private AI Foundation with NVIDIA

When you deploy a deep learning VM in vSphere IaaS control plane by using `kubectl` or directly on a vSphere cluster, you must fill in custom VM properties.

For information about deep learning VM images in VMware Private AI Foundation with NVIDIA, see About Deep Learning VM Images in VMware Private AI Foundation with NVIDIA.

OVF Properties of Deep Learning VMs

When you deploy a deep learning VM, you must fill in custom VM properties to automate the configuration of the Linux operating system, the deployment of the vGPU guest driver, and the deployment and configuration of NGC containers for the DL workloads.

The latest deep learning VM image has the following OVF properties:

Category	Parameter	Label in the vSphere Client	Description
Base OS Properties	instance-id	Instance ID	Required. A unique instance ID for the VM instance. An instance ID uniquely identifies an instance. When an instance ID changes, cloud-init treats the instance as a new instance and runs the cloud-init process to again.
	hostname	Hostname	Required. The host name of the appliance.
	seedfrom	URL to seed instance data from	Optional. An URL to pull the value for the user-data parameter and metadata from.
	public-keys	SSH public key	If provided, the instance populates the default user's SSH `authorized_keys` with this value.
	user-data	Encoded user-data	A set of scripts or other metadata that is inserted into the VM at provisioning time. This property is the actual contents of the `cloud-init` script. This value must be base64 encoded. ■ You can use this property to specify the DL workload container you want to deploy, such as PyTorch or TensorFlow. See Deep Learning Workloads in VMware Private AI Foundation with NVIDIA. ■ You use this property to set a static IP address to a virtual machine that is deployed directly on a vSphere cluster. See Assign a Static IP Address to a Deep Learning VM in VMware Private AI Foundation with NVIDIA.

Category	Parameter	Label in the vSphere Client	Description
	password	Default user's password	Required. The password for the default **vmware** user account.
vGPU Driver Installation	vgpu-license	vGPU license	Required. The NVIDIA vGPU client configuration token. The token is saved in the `/etc/nvidia/ClientConfigToken/client_configuration_token.tok` file.
	nvidia-portal-api-key	NVIDIA Portal API key	Required in a connected environment. The API key you downloaded from the NVIDIA Licensing Portal. The key is required for vGPU guest driver installation.
	vgpu-fallback-version	vGPU host driver version	Install directly this version of the vGPU guest driver.
	vgpu-url	URL for air-gapped vGPU downloads	Required in a disconnected environment. The URL to download the vGPU guest driver from. For information on the required configuration of the local Web server, see Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment.
DL Workload Automation	registry-uri	Registry URI	Required in a disconnected environment or if you plan to use a private container registry to avoid downloading images from the Internet. The URI of a private container registry with the deep learning workload container images. Required if you are referring to a private registry in `user-data` or `image-oneliner`.
	registry-user	Registry username	Required if you are using a private container registry that requires basic authentication.
	registry-passwd	Registry password	Required if you are using a private container registry that requires basic authentication.
	registry-2-uri	Secondary registry URI	Required if you are using a second private container registry that is based on Docker and requires basic authentication. For example, when deploying a deep learning VM with the NVIDIA RAG DL workload pre-installed, a pgvector image is downloaded from Docker Hub. You can use the `registry-2-` parameters to work around a pull rate limit for docker.io.
	registry-2-user	Secondary registry username	Required if you are using a second private container registry.
	registry-2-passwd	Secondary registry password	Required if you are using a second private container registry.
	image-oneliner	Encoded one-line command	A one-line bash command that is run at VM provisioning. This value must be base64 encoded. You can use this property to specify the DL workload container you want to deploy, such as PyTorch or TensorFlow. See Deep Learning Workloads in VMware Private AI Foundation with NVIDIA. **Caution** Avoid using both `user-data` and `image-oneliner`.

Category	Parameter	Label in the vSphere Client	Description
	docker-compose-uri	Encoded Docker compose file	Required if you need a Docker compose file to start the DL workload container. The contents of the `docker-compose.yaml` file that will be inserted into the virtual machine at provisioning after the virtual machine is started with GPU enabled. This value must be base64 encoded.
	config-json	Encoded config.json	The contents of a configuration file for adding details for proxy servers. This value must be base64 encoded. See Configure a Deep Learning VM with a Proxy Server.
	conda-environment-install	Conda Environment Install	A comma-separated list of Conda environments to be automatically installed after VM deployment is complete. Available environments: `pytorch2.3_py3.12`

Deep Learning Workloads in VMware Private AI Foundation with NVIDIA

You can provision a deep learning virtual machine with a supported deep learning (DL) workload in addition to its embedded components. The DL workloads are downloaded from the NVIDIA NGC catalog and are GPU-optimized and validated by NVIDIA and VMware by Broadcom.

For an overview of the deep learning VM images, see About Deep Learning VM Images in VMware Private AI Foundation with NVIDIA.

CUDA Sample

You can use a deep learning VM with running CUDA samples to explore vector addition, gravitational n-body simulation, or other examples on a VM. See the CUDA Samples page.

After the deep learning VM is launched, it runs a CUDA sample workload to test the vGPU guest driver. You can examine the test output in the `/var/log/dl.log` file.

Table 3-1. CUDA Sample Container Image

Component	Description
Container image	`nvcr.io/nvidia/k8s/cuda-sample:ngc_image_tag` For example: `nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8` For information on the CUDA Sample container images that are supported for deep learning VMs, see VMware Deep Learning VM Release Notes.
Required inputs	To deploy a CUDA Sample workload, you must set the OVF properties for the deep learning virtual machine in the following way: ■ Use one of the following properties that are specific for the CUDA Sample image. ■ Cloud-init script. Encode it in base64 format.

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

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

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

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

    docker run -d $REGISTRY_URI_PATH/nvidia/k8s/cuda-
sample:ngc_image_tag
- path: /opt/dlvm/utils.sh
```

Table 3-1. CUDA Sample Container Image (continued)

Component	Description

```
  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
```

Table 3-1. CUDA Sample Container Image (continued)

Component	Description
	(code continued below)

```
        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"
    }
```

For example, for vectoradd-cuda11.7.1-ubi8, provide the following script in base64 format:


```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICBz
ZXRfcHJveHkgImh0dHAiICJodHRwcyIgInNvY2tzNSIKICAgIHRyYXAgJ2Vycm9yX2V4
aXQgIlVuZXhwZWN0ZWQgZXJyb3Igb2NjdXJzIGF0IGRsIHdvcmtsb2FkIicgRVJSCiAg
ICBERUZBVUxUX1JFR19VUkk9Im52Y3IuaW8iCiAgICBSRUdJU1RSWV9VUklfUEFUSD0k
KGdyZXAgcmVnaXN0cnktdXJpIC9vcHQvZGx2bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAn
cy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKCiAgICBpZiBbWyAteiAiJFJF
R0lTVFJZX1VSSV9QQVRIIiBdXTsgdGhlbgogICAgICAjIElmIFJFR0lTVFJZX1VSSV9Q
QVRIIGlzIG51bGwgb3IgZW1wdHksIHVzZSB0aGUgZGVmYXVsdCB2YWx1ZQogICAgICBS
RUdJU1RSWV9VUklfUEFUSD0kREVGQVVMVF9SRUdfVVJJCiAgICAgIGVjaG8gIlJFR0lT
VFJZX1VSSV9QQVRIIHdhcyBlbXB0eS4gVXNpbmcgZGVmYXVsdDogJFJFR0lTVFJZX1VS
SV9QQVRIIgogICAgZmkKICAgIAogICAgIyBJZiBSRUdJU1RSWV9VUklfUEFUSCBjb250
YWlucyAnLycsIGV4dHJhY3QgdGhlIFVSSSBwYXJ0CiAgICBpZiBbWyAkUkVHSVNUUllf
VVJJX1BBVEggPT0gKiIvIiogXV07IHRoZW4KICAgICAgUkVHSVNUUllfVVJJPSQoZWNo
byAiJFJFR0lTVFJZX1VSSV9QQVRIIiB8IGN1dCAtZCcvJyAtZjEpCiAgICBlbHNlCiAg
ICAgIFJFR0lTVFJZX1VSST0kUkVHSVNUUllfVVJJX1BBVEgKICAgIGZpCiAgCiAgICBS
RUdJU1RSWV9VU0VSTkFNRT0kKGdyZXAgcmVnaXN0cnktdXNlciAvb3B0L2Rsdm0vb3Zm
LWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcp
CiAgICBSRUdJU1RSWV9QQVNTV09SRD0kKGdyZXAgcmVnaXN0cnktcGFzc3dkIC9vcHQv
ZGx2bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCku
Ki9cMS9wJykKICAgIGlmIFtbIC1uICIkUkVHSVNUUllfVVNFUk5BTUUiICYmIC1uICIk
UkVHSVNUUllfUEFTU1dPUkQiIF1dOyB0aGVuCiAgICAgIGRvY2tlciBsb2dpbiAtdSAk
UkVHSVNUUllfVVNFUk5BTUUgLXAgJFJFR0lTVFJZX1BBU1NXT1JEICRSRUdJU1RSWV9V
UkkKICAgIGVsc2UKICAgICAgZWNobyAiV2FybmluZzogdGhlIHJlZ2lzdHJ5J3MgdXNl
cm5hbWUgYW5kIHBhc3N3b3JkIGFyZSBpbnZhbGlkLCBTa2lwcGluZyBEb2NrZXIgbG9n
aW4uIgogICAgZmkKICAgIAogICAgZG9ja2VyIHJ1biAtZCAkUkVHSVNUUllfVVJJX1BB
VEgvbnZpZGlhL2s4cy9jdWRhLXNhbXBsZTp2ZWN0b3JhZGQtY3VkYTExLjcuMS11Ymk4
CgotIHBhdGg6IC9vcHQvZGx2bS91dGlscy5zaAogIHBlcm1pc3Npb25zOiAnMDc1NScK
ICBjb250ZW50OiB8CiAgICAjIS9iaW4vYmFzaAogICAgZXJyb3JfZXhpdCgpIHsKICAg
ICAgZWNobyAiRXJyb3I6ICQxIiA+JjIKICAgICAgdm10b29sc2QgLS1jbWQgImluZm8t
c2V0IGd1ZXN0aW5mby52bXNlcnZpY2UuYm9vdHN0cmFwLmNvbmRpdGlvbiBmYWxzZSwg
RExXb3JrbG9hZEZhaWx1cmUsICQxIgogICAgICBleGl0IDEKICAgIH0KCiAgICBjaGVj
a19wcm90b2NvbCgpIHsKICAgICAgbG9jYWwgcHJveHlfdXJsPSQxCiAgICAgIHNoaWZ0
CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIpCiAgICAgIGlmIFtb
```

Table 3-1. CUDA Sample Container Image (continued)

Component	Description


```
IC1uICIke3Byb3h5X3VybH0iIF1dOyB0aGVuCiAgICAgICAgbG9jYWwgcHJvdG9jb2w9
JChlY2hvICIke3Byb3h5X3VybH0iIHwgYXdrIC1GICc6Ly8nICd7aWYgKE5GID4gMSkg
cHJpbnQgJDE7IGVsc2UgcHJpbnQgIiJ9JykKICAgICAgICBpZiBbIC16ICIkcHJvdG9j
b2wiIF07IHRoZW4KICAgICAgICAgIGVjaG8gIk5vIHNwZWNpZmljIHByb3RvY29sIHBy
b3ZpZGVkLiBTa2lwcGluZyBwcm90b2NvbCBjaGVjay4iCiAgICAgICAgICByZXR1cm4g
MAogICAgICAgIGZpCiAgICAgICAgbG9jYWwgcHJvdG9jb2xfaW5jbHVkZWQ9ZmFsc2UK
ICAgICAgICBmb3IgdmFyIGluICIke3N1cHBvcnRlZF9wcm90b2NvbHNbQF19IjsgZG8K
ICAgICAgICAgIGlmIFtbICIke3Byb3RvY29sfSIgPT0gIiR7dmFyfSIgXV07IHRoZW4K
ICAgICAgICAgICAgcHJvdG9jb2xfaW5jbHVkZWQ9dHJ1ZQogICAgICAgICAgICBicmVh
awogICAgICAgICAgZmkKICAgICAgICBkb25lCiAgICAgICAgaWYgW1sgIiR7cHJvdG9j
b2xfaW5jbHVkZWR9IiA9PSBmYWxzZSBdXTsgdGhlbgogICAgICAgICAgZXJyb3JfZXhp
dCAiVW5zdXBwb3J0ZWQgcHJvdG9jb2w6ICR7cHJvdG9jb2x9LiBTdXBwb3J0ZWQgcHJv
dG9jb2xzIGFyZTogJHtzdXBwb3J0ZWRfcHJvdG9jb2xzWypdfSIKICAgICAgICBmaQog
ICAgICBmaQogICAgfQoKICAgICMgJEA6IGxpc3Qgb2Ygc3VwcG9ydGVkIHByb3RvY29s
cwogICAgc2V0X3Byb3h5KCkgewogICAgICBsb2NhbCBzdXBwb3J0ZWRfcHJvdG9jb2xz
PSgiJEAiKQoKICAgICAgQ09ORklHX0pTT05fQkFTRTY0PSQoZ3JlcCAnY29uZmlnLWpz
b24nIC9vcHQvZGx2bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJc
KFteIl0qXCkuKi9cMS9wJykKICAgICAgQ09ORklHX0pTT049JChlY2hvICR7Q09ORklH
X0pTT05fQkFTRTY0fSB8IGJhc2U2NCAtLWRlY29kZSkKCiAgICAgIEhUVFBfUFJPWFlf
VVJMPSQoZWNobyAiJHtDT05GSUdfSlNPTn0iIHwganEgLXIgJy5odHRwX3Byb3h5IC8v
IGVtcHR5JykKICAgICAgSFRUUFNfUFJPWFlfVVJMPSQoZWNobyAiJHtDT05GSUdfSlNP
Tn0iIHwganEgLXIgJy5odHRwc19wcm94eSAvLyBlbXB0eScpCiAgICAgIGlmIFtbICQ/
IC1uZSAwIHx8ICgteiAiJHtIVFRQX1BST1hZX1VSTH0iICYmIC16ICIke0hUVFBTX1BS
T1hZX1VSTH0iKSBdXTsgdGhlbgogICAgICAgIGVjaG8gIkluZm86IFRoZSBjb25maWct
anNvbiB3YXMgcGFyc2VkLCBidXQgbm8gcHJveHkgc2V0dGluZ3Mgd2VyZSBmb3VuZC4i
CiAgICAgICAgcmV0dXJuIDAKICAgICAgZmkKICAKICAgICAgY2hlY2tfcHJvdG9jb2wg
IiR7SFRUUF9QUk9YWV9VUkx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAg
ICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVk
X3Byb3RvY29sc1tAXX0iCgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9l
dGMvZW52aXJvbm1lbnQ7IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94
eT0ke0hUVFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hU
VFBTX1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJP
WFlfVVJMfQogICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VS
TH0KICAgICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4g
L2V0Yy9lbnZpcm9ubWVudAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAg
ICAgIGZpCiAgICAgIAogICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJv
eHkKICAgICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2Vydmlj
ZS5kCiAgICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBf
UFJPWFk9JHtIVFRQX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBT
X1BST1hZPSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25tZW50PVwiTk9f
UFJPWFk9bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0v
ZG9ja2VyLnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgIHN5c3RlbWN0bCBkYWVtb24t
cmVsb2FkCiAgICAgIHN5c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAi
SW5mbzogZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1
cmVkIHRvIHVzZSB0aGUgcHJveHkgc2V0dGluZ3MiCiAgICB9
```


which corresponds to the following script in plain-text format:

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

Table 3-1. CUDA Sample Container Image (continued)

Component	Description

```
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
```

Table 3-1. CUDA Sample Container Image (continued)

Component	Description
	(code continued below)

```
          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"
    }
```

- Image one-liner. Encode it in base64 format

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

Table 3-1. CUDA Sample Container Image (continued)

Component	Description
	For example, for vectoradd-cuda11.7.1-ubi8, provide the following script in base64 format: `ZG9ja2VyIHJ1biAtZCBudmNyLmlvL252aWRpYS9rOHMvY3VkYS1zYW1wbGU6dmVjdG9yYWRkLWN1ZGExMS43LjEtdWJpOA==` which corresponds to the following script in plain-text format: `docker run -d nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8` ■ Enter the vGPU guest driver installation properties, such as `vgpu-license` and `nvidia-portal-api-key`. ■ Provide values for the properties required for a disconnected environment as needed. See OVF Properties of Deep Learning VMs.
Output	■ Installation logs for the vGPU guest driver in `/var/log/vgpu-install.log`. To verify that the vGPU guest driver is installed, and the license is allocated, run the following command: `nvidia-smi -q |grep -i license` ■ Cloud-init script logs in `/var/log/dl.log`.

PyTorch

You can use a deep learning VM with a PyTorch library to explore conversational AI, NLP, and other types of AI models, on a VM. See the PyTorch page.

After the deep learning VM is launched, it starts a JupyterLab instance with PyTorch packages installed and configured.

Table 3-2. PyTorch Container Image

<table>
<tr><th>Component</th><th>Description</th></tr>
<tr><td>Container image</td><td><pre>nvcr.io/nvidia/pytorch:ngc_image_tag</pre>
For example:
<pre>nvcr.io/nvidia/pytorch:23.10-py3</pre>
For information on the PyTorch container images that are supported for deep learning VMs, see VMware Deep Learning VM Release Notes.</td></tr>
<tr><td>Required inputs</td><td>To deploy a PyTorch workload, you must set the OVF properties for the deep learning virtual machine in the following way:
Use one of the following properties that are specific for the PyTorch image.
Cloud-init script. Encode it in base64 format.
<pre>#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-</pre></td></tr>
</table>

Table 3-2. PyTorch Container Image (continued)

Component	Description

```
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[@]}"
```

Table 3-2. PyTorch Container Image (continued)

Component	Description
	`` 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" } ``` For example, for pytorch:23.10-py3, provide the following script in base 64 format: I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0 cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jr bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCgog ICAgREVGQVVMVF9SRUdfVVJJPSJudmNyLmlvIgogICAgUkVHSVNUUllfVVJJX1BBVEg9 JChncmVwIHJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4g J3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCgogICAgaWYgW1sgLXogIiRS RUdJU1RSWV9VUklfUEFUSCIgXV07IHRoZW4KICAgICAgIyBJZiBSRUdJU1RSWV9VUklf UEFUSCBpcyBudWxsIG9yIGVtcHR5LCB1c2UgdGhlIGRlZmF1bHQgdmFsdWUKICAgICAg UkVHSVNUUllfVVJJX1BBVEg9JERFRkFVTFRfUkVHX1VSSQogICAgICBlY2hvICJSRUdJ U1RSWV9VUklfUEFUSCB3YXMgZW1wdHkuIFVzaW5nIGRlZmF1bHQ6ICRSRUdJU1RSWV9V UklfUEFUSCIKICAgIGZpCiAgICAKICAgICMgSWYgUkVHSVNUUllfVVJJX1BBVEggY29u dGFpbnMgJy8nLCBleHRyYWN0IHRoZSBVUkkgcGFydAogICAgaWYgW1sgJFJFR0lTVFJZ X1VSSV9QQVRIID09ICoiLyIqIF1dOyB0aGVuCiAgICAgIFJFR0lTVFJZX1VSST0kKGVj aG8gIiRSRUdJU1RSWV9VUklfUEFUSCIgfCBjdXQgLWQnLycgLWYxKQogICAgZWxzZQog ICAgICBSRUdJU1RSWV9VUkk9JFJFR0lTVFJZX1VSSV9QQVRICiAgICBmaQogIAogICAg UkVHSVNUUllfVVNFUk5BTUU9JChncmVwIHJlZ2lzdHJ5LXVzZXIgL29wdC9kbHZtL292 Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3An KQogICAgUkVHSVNUUllfUEFTU1dPUkQ9JChncmVwIHJlZ2lzdHJ5LXBhc3N3ZCAvb3B0 L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwp LiovXDEvcCcpCiAgICBpZiBbWyAtbiAiJFJFR0lTVFJZX1VTRVJOQU1FIiAmJiAtbiAi JFJFR0lTVFJZX1BBU1NXT1JEIiBdXTsgdGhlbgogICAgICBkb2NrZXIgbG9naW4gLXUg JFJFR0lTVFJZX1VTRVJOQU1FIC1wICRSRUdJU1RSWV9QQVNTV09SRCAkUkVHSVNUUllf VVJJCiAgICBlbHNlCiAgICAgIGVjaG8gIldhcm5pbmc6IHRoZSByZWdpc3RyeSdzIHVz ZXJuYW1lIGFuZCBwYXNzd29yZCBhcmUgaW52YWxpZCwgU2tpcHBpbmcgRG9ja2VyIGxv Z2luLiIKICAgIGZpCgogICAgZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC1wIDg4ODg6 ODg4OCAkUkVHSVNUUllfVVJJX1BBVEgvbnZpZGlhL3B5dG9yY2g6MjMuMTAtcHkzIC91 c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9vdCAtLWlwPSogLS1wb3J0 PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAudG9rZW49JycgLS1Ob3RlYm9v a0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGlyPS93b3Jrc3BhY2UKCi0g cGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNv

Table 3-2. PyTorch Container Image (continued)

Component	Description


```
bnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAgICBl
Y2hvICJFcnJvcjogJDEiID4mMgogICAgICB2bXRvb2xzZCAtLWNtZCAiaW5mby1zZXQg
Z3Vlc3RpbmZvLnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIGZhbHNlLCBETFdv
cmtsb2FkRmFpbHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgfQoKICAgIGNoZWNrX3By
b3RvY29sKCkgewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQKICAg
ICAgbG9jYWwgc3VwcG9ydGVkX3Byb3RvY29scz0oIiRAIikKICAgICAgaWYgW1sgLW4g
IiR7cHJveHlfdXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm90b2NvbD0kKGVj
aG8gIiR7cHJveHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBwcmlu
dCAkMTsgZWxzZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2NvbCIg
XTsgdGhlbgogICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMgcHJvdG9jb2wgcHJvdmlk
ZWQuIFNraXBwaW5nIHByb3RvY29sIGNoZWNrLiIKICAgICAgICAgIHJldHVybiAwCiAg
ICAgICAgZmkKICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQogICAg
ICAgIGZvciB2YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwogICAg
ICAgICAgaWYgW1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9IiBdXTsgdGhlbgogICAg
ICAgICAgICBwcm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgIGJyZWFrCiAg
ICAgICAgICBmaQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2NvbF9p
bmNsdWRlZH0iID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0ICJV
bnN1cHBvcnRlZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1cHBvcnRlZCBwcm90b2Nv
bHMgYXJlOiAke3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgICAgZmkKICAg
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
```


which corresponds to the following script in plain-text format.

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

Table 3-2. PyTorch Container Image (continued)

Component	Description

```
    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
```

Table 3-2. PyTorch Container Image (continued)

Component	Description

```
          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
```

Table 3-2. PyTorch Container Image (continued)

<table>
<tr><th>Component</th><th>Description</th></tr>
<tr><td></td><td>
<pre>   echo "Info: docker and system environment are now configured
to use the proxy settings"
    }</pre>

Image one-liner. Encode it in base64 format.
<pre>docker run -d -p 8888:8888 nvcr.io/nvidia/pytorch:ngc_image_tag
/usr/local/bin/jupyter lab --allow-root --ip=* --port=8888 --
no-browser --NotebookApp.token='' --NotebookApp.allow_origin='*' --
notebook-dir=/workspace</pre>
For example, for pytorch:23.10-py3, provide the following script in base 64 format:
<pre>ZG9ja2VyIHJ1biAtZCAtcCA4ODg4Ojg4ODggbnZjci5pby9udmlkaWEvcHl0b3JjaDoy
My4xMC1weTMgL3Vzci9sb2NhbC9iaW4vanVweXRlciBsYWIgLS1hbGxvdy1yb290IC0t
aXA9KiAtLXBvcnQ9ODg4OCAtLW5vLWJyb3dzZXIgLS1Ob3RlYm9va0FwcC50b2tlbj0n
JyAtLU5vdGVib29rQXBwLmFsbG93X29yaWdpbj0nKicgLS1ub3RlYm9vay1kaXI9L3dv
cmtzcGFjZQ==</pre>
which corresponds to the following script in plain-text format:
<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>

Enter the vGPU guest driver installation properties, such as <code>vgpu-license</code> and <code>nvidia-portal-api-key</code>.
Provide values for the properties required for a disconnected environment as needed.

See OVF Properties of Deep Learning VMs.
</td></tr>
<tr><td>Output</td><td>

Installation logs for the vGPU guest driver in <code>/var/log/vgpu-install.log</code>.

To verify that the vGPU guest driver is installed, run the <code>nvidia-smi</code> command.
Cloud-init script logs in <code>/var/log/dl.log</code>.
PyTorch container.

To verify that the PyTorch container is running, run the <code>sudo docker ps -a</code> and <code>sudo docker logs container_id</code> command.
JupyterLab instance that you can access at <code>http://dl_vm_ip:8888</code>

In the terminal of JupyterLab, verify that the following functionality is available in the notebook:

To verify that JupyterLab can access the vGPU resource, run <code>nvidia-smi</code>.
To verify that the PyTorch related packages are installed, run <code>pip show</code>.

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

TensorFlow

You can use a deep learning VM with a TensorFlow library to explore conversational AI, NLP, and other types of AI models, on a VM. See the TensorFlow page.

After the deep learning VM is launched, it starts a JupyterLab instance with TensorFlow packages installed and configured.

Table 3-3. TensorFlow Container Image

<table>
<tr><th>Component</th><th>Description</th></tr>
<tr><td>Container image</td><td>
<pre>nvcr.io/nvidia/tensorflow:ngc_image_tag</pre>
For example:
<pre>nvcr.io/nvidia/tensorflow:23.10-tf2-py3</pre>
For information on the TensorFlow container images that are supported for deep learning VMs, see VMware Deep Learning VM Release Notes.
</td></tr>
<tr><td>Required inputs</td><td>
To deploy a TensorFlow workload, you must set the OVF properties for the deep learning virtual machine in the following way:
Use one of the following properties that are specific for the TensorFlow image.
Cloud-init script. Encode it in base64 format.
<pre>
#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-
</pre>

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

Table 3-3. TensorFlow Container Image (continued)

Component	Description
	(code continued below)

```
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[@]}"
```

Table 3-3. TensorFlow Container Image (continued)

Component	Description
	`` 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" } ``` For example, for tensorflow:23.10-tf2-py3, provide the following script in base64 format: I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0 cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jr bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCiAg ICAKICAgIERFRkFVTFRfUkVHX1VSST0ibnZjci5pbyIKICAgIFJFR0lTVFJZX1VSSV9Q QVRIPSQoZ3JlcCByZWdpc3RyeS11cmkgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2Vk IC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIGlmIFtbIC16 ICIkUkVHSVNUUllfVVJJX1BBVEgiIF1dOyB0aGVuCiAgICAgICMgSWYgUkVHSVNUUllf VVJJX1BBVEggaXMgbnVsbCBvciBlbXB0eSwgdXNlIHRoZSBkZWZhdWx0IHZhbHVlCiAg ICAgIFJFR0lTVFJZX1VSSV9QQVRIPSRERUZBVUxUX1JFR19VUkkKICAgICAgZWNobyAi UkVHSVNUUllfVVJJX1BBVEggd2FzIGVtcHR5LiBVc2luZyBkZWZhdWx0OiAkUkVHSVNU UllfVVJJX1BBVEgiCiAgICBmaQogICAgCiAgICAjIElmIFJFR0lTVFJZX1VSSV9QQVRI IGNvbnRhaW5zICcvJywgZXh0cmFjdCB0aGUgVVJJIHBhcnQKICAgIGlmIFtbICRSRUdJ U1RSWV9VUklfUEFUSCA9PSAqIi8iKiBdXTsgdGhlbgogICAgICBSRUdJU1RSWV9VUkk9 JChlY2hvICIkUkVHSVNUUllfVVJJX1BBVEgiIHwgY3V0IC1kJy8nIC1mMSkKICAgIGVs c2UKICAgICAgUkVHSVNUUllfVVJJPSRSRUdJU1RSWV9VUklfUEFUSAogICAgZmkKICAK ICAgIFJFR0lTVFJZX1VTRVJOQU1FPSQoZ3JlcCByZWdpc3RyeS11c2VyIC9vcHQvZGx2 bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9c MS9wJykKICAgIFJFR0lTVFJZX1BBU1NXT1JEPSQoZ3JlcCByZWdpc3RyeS1wYXNzd2Qg L29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14i XSpcKS4qL1wxL3AnKQogICAgaWYgW1sgLW4gIiRSRUdJU1RSWV9VU0VSTkFNRSIgJiYg LW4gIiRSRUdJU1RSWV9QQVNTV09SRCIgXV07IHRoZW4KICAgICAgZG9ja2VyIGxvZ2lu IC11ICRSRUdJU1RSWV9VU0VSTkFNRSAtcCAkUkVHSVNUUllfUEFTU1dPUkQgJFJFR0lT VFJZX1VSSQogICAgZWxzZQogICAgICBlY2hvICJXYXJuaW5nOiB0aGUgcmVnaXN0cnkn cyB1c2VybmFtZSBhbmQgcGFzc3dvcmQgYXJlIGludmFsaWQsIFNraXBwaW5nIERvY2tl ciBsb2dpbi4iCiAgICBmaQogICAgCiAgICBkb2NrZXIgcnVuIC1kIC0tZ3B1cyBhbGwg LXAgODg4ODo4ODg4ICRSRUdJU1RSWV9VUklfUEFUSC9udmlkaWEvdGVuc29yZmxvdzoy My4xMC10ZjItcHkzIC91c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ctcm9v dCAtLWlwPSogLS1wb3J0PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAudG9r ZW49JycgLS1Ob3RlYm9va0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2stZGly PS93b3Jrc3BhY2UKCi0gcGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lv

Table 3-3. TensorFlow Container Image (continued)

Component	Description


```
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
```


which corresponds to the following script in plain-text format:

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

Table 3-3. TensorFlow Container Image (continued)

Component	Description

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

Table 3-3. TensorFlow Container Image (continued)

Component	Description

```
          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
```

Table 3-3. TensorFlow Container Image (continued)

<table>
<tr><th>Component</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
      echo "Info: docker and system environment are now configured
to use the proxy settings"
    }
</pre>
Image one-liner. Encode it in base64 format.
<pre>
docker run -d -p 8888:8888 nvcr.io/nvidia/tensorflow:ngc_image_tag
/usr/local/bin/jupyter lab --allow-root --ip=* --port=8888 --
no-browser --NotebookApp.token='' --NotebookApp.allow_origin='*' --
notebook-dir=/workspace
</pre>
For example, for tensorflow:23.10-tf2-py3, provide the following script in base64 format:
ZG9ja2VyIHJ1biAtZCAtcCA4ODg4Ojg4ODggbnZjci5pby9udmlkaWEvdGVuc29yZmxv
dzoyMy4xMC10ZjItcHkzIC91c3IvbG9jYWwvYmluL2p1cHl0ZXIgbGFiIC0tYWxsb3ct
cm9vdCAtLWlwPSogLS1wb3J0PTg4ODggLS1uby1icm93c2VyIC0tTm90ZWJvb2tBcHAu
dG9rZW49JycgLS1Ob3RlYm9va0FwcC5hbGxvd19vcmlnaW49JyonIC0tbm90ZWJvb2st
ZGlyPS93b3Jrc3BhY2U=
which corresponds to the following script in plain-text format:
<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>
Enter the vGPU guest driver installation properties, such as <code>vgpu-license</code> and <code>nvidia-portal-api-key</code>.
Provide values for the properties required for a disconnected environment as needed.
See OVF Properties of Deep Learning VMs.</td></tr>
<tr><td>Output</td><td>Installation logs for the vGPU guest driver in <code>/var/log/vgpu-install.log</code>.
To verify that the vGPU guest driver is installed, log in to the VM over SSH and run the <code>nvidia-smi</code> command.
Cloud-init script logs in <code>/var/log/dl.log</code>.
TensorFlow container.
To verify that the TensorFlow container is running, run the <code>sudo docker ps -a</code> and <code>sudo docker logs container_id</code> commands.
JupyterLab instance that you can access at <code>http://dl_vm_ip:8888</code>.
In the terminal of JupyterLab, verify that the following functionality is available in the notebook:
To verify that JupyterLab can access the vGPU resource, run <code>nvidia-smi</code>.
To verify that the TensorFlow related packages are installed, run <code>pip show</code>.</td></tr>
</table>

DCGM Exporter

You can use a deep learning VM with a Data Center GPU Manager (DCGM) exporter to monitor the health of and get metrics from GPUs used by a DL workload, using NVIDIA DCGM, Prometheus, and Grafana.

See the DCGM Exporter page.

In a deep learning VM, you run the DCGM Exporter container together with a DL workload that performs AI operations. After the deep learning VM is started, DCGM Exporter is ready to collect vGPU metrics and export the data to another application for further monitoring and visualization. You can run the monitored DL workload as a part of the cloud-init process or from the command line after the virtual machine is started.

Table 3-4. DCGM Exporter Container Image

<table>
<tr><th>Component</th><th>Description</th></tr>
<tr><td>Container image</td><td>
<pre><code>nvcr.io/nvidia/k8s/dcgm-exporter:ngc_image_tag</code></pre>
For example:
<pre><code>nvcr.io/nvidia/k8s/dcgm-exporter:3.2.5-3.1.8-ubuntu22.04</code></pre>
For information on the DCGM Exporter container images that are supported for deep learning VMs, see VMware Deep Learning VM Release Notes.
</td></tr>
<tr><td>Required inputs</td><td>
To deploy a DCGM Exporter workload, you must set the OVF properties for the deep learning virtual machine in the following way:

Use one of the following properties that are specific for the DCGM Exporter image.

Cloud-init script. Encode it in base64 format.
<pre><code>#cloud-config
write_files:
- path: /opt/dlvm/dl_app.sh
  permissions: '0755'
  content: |
    #!/bin/bash
    set -eu
    source /opt/dlvm/utils.sh
    trap 'error_exit "Unexpected error occurs at dl workload"' ERR
    set_proxy "http" "https" "socks5"

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

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

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

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

    docker run -d --gpus all --cap-add SYS_ADMIN --rm -p 9400:9400
$REGISTRY_URI_PATH/nvidia/k8s/dcgm-exporter:ngc_image_tag</code></pre>

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

Table 3-4. DCGM Exporter Container Image (continued)

Component	Description

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

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

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

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

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

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

Table 3-4. DCGM Exporter Container Image (continued)

Component	Description
	`` if ! grep -q 'http_proxy' /etc/environment; then echo "export http_proxy=${HTTP_PROXY_URL} export https_proxy=${HTTPS_PROXY_URL} export HTTP_PROXY=${HTTP_PROXY_URL} export HTTPS_PROXY=${HTTPS_PROXY_URL} export no_proxy=localhost,127.0.0.1" >> /etc/environment source /etc/environment fi # Configure Docker to use a proxy mkdir -p /etc/systemd/system/docker.service.d echo "[Service] Environment=\"HTTP_PROXY=${HTTP_PROXY_URL}\" Environment=\"HTTPS_PROXY=${HTTPS_PROXY_URL}\" Environment=\"NO_PROXY=localhost,127.0.0.1\"" > /etc/systemd/ system/docker.service.d/proxy.conf systemctl daemon-reload systemctl restart docker echo "Info: docker and system environment are now configured to use the proxy settings" } ``` For example, for a deep learning VM with a pre-installed a dcgm-exporter:3.2.5-3.1.8-ubuntu22.04 DCGM Exporter instance, provide the following script in base64 format I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0 cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jr bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCiAg ICAKICAgIERFRkFVTFRfUkVHX1VSST0ibnZjci5pbyIKICAgIFJFR0lTVFJZX1VSSV9Q QVRIPSQoZ3JlcCByZWdpc3RyeS11cmkgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2Vk IC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIGlmIFtbIC16 ICIkUkVHSVNUUllfVVJJX1BBVEgiIF1dOyB0aGVuCiAgICAgICMgSWYgUkVHSVNUUllf VVJJX1BBVEggaXMgbnVsbCBvciBlbXB0eSwgdXNlIHRoZSBkZWZhdWx0IHZhbHVlCiAg ICAgIFJFR0lTVFJZX1VSSV9QQVRIPSRERUZBVUxUX1JFR19VUkkKICAgICAgZWNobyAi UkVHSVNUUllfVVJJX1BBVEggd2FzIGVtcHR5LiBVc2luZyBkZWZhdWx0OiAkUkVHSVNU UllfVVJJX1BBVEgiCiAgICBmaQogICAgCiAgICAjIElmIFJFR0lTVFJZX1VSSV9QQVRI IGNvbnRhaW5zICcvJywgZXh0cmFjdCB0aGUgVVJJIHBhcnQKICAgIGlmIFtbICRSRUdJ U1RSWV9VUklfUEFUSCA9PSAqIi8iKiBdXTsgdGhlbgogICAgICBSRUdJU1RSWV9VUkk9 JChlY2hvICIkUkVHSVNUUllfVVJJX1BBVEgiIHwgY3V0IC1kJy8nIC1mMSkKICAgIGVs c2UKICAgICAgUkVHSVNUUllfVVJJPSRSRUdJU1RSWV9VUklfUEFUSAogICAgZmkKICAK ICAgIFJFR0lTVFJZX1VTRVJOQU1FPSQoZ3JlcCByZWdpc3RyeS11c2VyIC9vcHQvZGx2 bS9vdmYtZW52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9c MS9wJykKICAgIFJFR0lTVFJZX1BBU1NXT1JEPSQoZ3JlcCByZWdpc3RyeS1wYXNzd2Qg L29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14i XSpcKS4qL1wxL3AnKQogICAgaWYgW1sgLW4gIiRSRUdJU1RSWV9VU0VSTkFNRSIgJiYg LW4gIiRSRUdJU1RSWV9QQVNTV09SRCIgXV07IHRoZW4KICAgICAgZG9ja2VyIGxvZ2lu IC1lICRSRUdJU1RSWV9VU0VSTkFNRSAtcCAkUkVHSVNUUllfUEFTU1dPUkQgJFJFR0lT VFJZX1VSSQogICAgZWxzZQogICAgICBlY2hvICJXYXJuaW5nOiB0aGUgcmVnaXN0cnkn cyB1c2VybmFtZSBhbmQgcGFzc3dvcmQgYXJlIGludmFsaWQsIFNraXBwaW5nIERvY2tl ciBsb2dpbi4iCiAgICBmaQoKICAgIGRvY2tlciBydW4gLWQgLS1ncHVzIGFsbCAtLWNh cC1hZGQgU1lTX0FETUlOIC0tcm0gLXAgOTQwMDo5NDAwICRSRUdJU1RSWV9VUklfUEFU SC9udmlkaWEvazhzL2RjZ20tZXhwb3J0ZXI6My4yLjUtMy4xLjgtdWJ1bnR1MjIuMDQK Ci0gcGF0aDogL29wdC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1Jwog IGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAg ICBlY2hvICJFcnJvcjogJDEiID4mMgogICAgICB2bXRvb2xzZCAtLWNtZCAiaW5mby1z ZXQgZ3Vlc3RpbmZvLnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIGZhbHNlLCBE

Table 3-4. DCGM Exporter Container Image (continued)

Component | **Description**


```
TFdvcmtsb2FkRmFpbHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgfQoKICAgIGNoZWNr
X3Byb3RvY29sKCkgewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQK
ICAgICAgbG9jYWwgc3VwcG9ydGVkX3Byb3RvY29scz0oIiRAIikKICAgICAgaWYgW1sg
LW4gIiR7cHJveHlfdXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm90b2NvbD0k
KGVjaG8gIiR7cHJveHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBw
cmludCAkMTsgZWxzZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2Nv
bCIgXTsgdGhlbgogICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMgcHJvdG9jb2wgcHJv
dmlkZWQuIFNraXBwaW5nIHByb3RvY29sIGNoZWNrLiIKICAgICAgICAgIHJldHVybiAw
CiAgICAgICAgZmkKICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQog
ICAgICAgIGZvciB2YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwog
ICAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9IiBdXTsgdGhlbgog
ICAgICAgICAgICBwcm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgIGJyZWFr
CiAgICAgICAgICBmaQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2Nv
bF9pbmNsdWRlZH0iID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0
ICJVbnN1cHBvcnRlZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1cHBvcnRlZCBwcm90
b2NvbHMgYXJlOiAke3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgIGZpCiAg
ICAgIGZpCiAgICB9CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xz
CiAgICBzZXRfcHJveHkoKSB7CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9
KCIkQCIpCgogICAgICBDT05GSUdfSlNPTl9CQVNFNjQ9JChncmVwICdjb25maWctanNv
bicgL29wdC9kbHZtL292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9Ilwo
W14iXSpcKS4qL1wxL3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdf
SlNPTl9CQVNFNjR9IHwgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgSFRUUF9QUk9YWV9V
Ukw9JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0dHBfcHJveHkgLy8g
ZW1wdHknKQogICAgICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09O
fSIgfCBqcSAtciAnLmh0dHBzX3Byb3h5IC8vIGVtcHR5JykKICAgICAgaWYgW1sgJD8g
LW5lIDAgfHwgKC16ICIke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJP
WFlfVVJMfSIpIF1dOyB0aGVuCiAgICAgICAgZWNobyAiSW5mbzogVGhlIGNvbmZpZy1q
c29uIHdhcyBwYXJzZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kLiIK
ICAgICAgICByZXR1cm4gMAogICAgICBmaQoKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7
SFRUUF9QUk9YWV9VUkx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAg
Y2hlY2tfcHJvdG9jb2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVkX3By
b3RvY29sc1tAXX0iCgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldGMv
ZW52aXJvbm1lbnQ7IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0k
e0hUVFBfUFJPWFlfVVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBT
X1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJPWFlf
VVJMfQogICAgICAgIGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH0K
ICAgICAgICBleHBvcnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0
Yy9lbnZpcm9ubWVudAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAgICAg
IGZpCiAgICAgIAogICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJveHkK
ICAgICAgbWtkaXIgLXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5k
CiAgICAgIGVjaG8gIltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJP
WFk9JHtIVFRQX1BST1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BS
T1hZPSR7SFRUUFNfUFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25tZW50PVwiTk9fUFJP
WFk9bG9jYWxob3N0LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9j
a2VyLnNlcnZpY2UuZC9wcm94eS5jb25mCiAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVs
b2FkCiAgICAgIHN5c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAiSW5m
bzogZG9ja2VyIGFuZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1cmVk
IHRvIHVzZSB0aGUgcHJveHkgc2V0dGluZ3MiCiAgICB9
```


which corresponds to the following script in plain-text format:

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

Table 3-4. DCGM Exporter Container Image (continued)

Component	Description

```
    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
        fi
```

Table 3-4. DCGM Exporter Container Image (continued)

Component	Description

```
        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
```

Table 3-4. DCGM Exporter Container Image (continued)

<table>
<tr><th>Component</th><th>Description</th></tr>
<tr><td></td><td>

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

Note You can also add the instructions for running the DL workload whose GPU performance you want to measure with DCGM Exporter to the cloud-init script.

- Image one-liner. Encode it in base64 format.

```
docker run -d --gpus all --cap-add SYS_ADMIN --rm -p 9400:9400
nvcr.io/nvidia/k8s/dcgm-exporter:ngc_image_tag-ubuntu22.04
```

For example, for dcgm-exporter:3.2.5-3.1.8-ubuntu22.04, provide the following script in base64 format:


```
ZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tY2FwLWFkZCBTWVNfQURNSU4gLS1ybSAt
cCA5NDAwOjk0MDAgbnZjci5pby9udmlkaWEvazhzL2RjZ20tZXhwb3J0ZXI6My4yLjUt
My4xLjgtdWJ1bnR1MjIuMDQ=
```


which corresponds to the following script in plain-text format:

```
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
```

- Enter the vGPU guest driver installation properties, such as `vgpu-license` and `nvidia-portal-api-key`.
- Provide values for the properties required for a disconnected environment as needed.

See OVF Properties of Deep Learning VMs.
</td></tr>
<tr><td>Output</td><td>

- Installation logs for the vGPU guest driver in `/var/log/vgpu-install.log`.

  To verify that the vGPU guest driver is installed, log in to the VM over SSH and run the `nvidia-smi` command.
- Cloud-init script logs in `/var/log/dl.log`.
- DCGM Exporter that you can access at http://*dl_vm_ip*:9400.

Next, in the deep learning VM, you run a DL workload, and visualize the data on another virtual machine by using Prometheus at http://*visualization_vm_ip*:9090 and Grafana at http://*visualization_vm_ip*:3000.
</td></tr>
</table>

Run a DL Workload on the Deep Leaning VM

Run the DL workload you want to collect vGPU metrics for and export the data to another application for further monitoring and visualization.

1 Log in to the deep learning VM as **vmware** over SSH.

2 Add the **vmware** user account to the **docker** group by running the following command.

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

3 Run the container for the DL workload, pulling it from the NVIDIA NGC catalog or from a local container registry.

For example, to run the following command to run the tensorflow:23.10-tf2-py3 image from 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 Start using the DL workload for AI development.

Install Prometheus and Grafana

You can visualize and monitor the vGPU metrics from the DCGM Exporter virtual machine on a virtual machine running Prometheus and Grafana.

1 Create a visualization VM with Docker Community Engine installed.

2 Connect to the VM over SSH and create a YAML file for 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 Create a data path.

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

4 Create a Docker compose file to install Prometheus and 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 Start the Prometheus and Grafana containers.

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

View vGPU Metrics in Prometheus

You can access Prometheus at `http://`*`visualization-vm-ip`*`:9090`. You can view the following vGPU information in the Prometheus UI:

Information	UI Section
Raw vGPU metrics from the deep learning VM	**Status > Target** To view the raw vGPU metrics from the deep learning VM, click the endpoint entry.
Graph expressions	1 On the main navigation bar, click the **Graph** tab. 2 Enter an expression and click **Execute**

For more information on using Prometheus, see the Prometheus documentation.

Visualize Metrics in Grafana

Set Prometheus as a data source for Grafana and visualize the vGPU metrics from the deep learning VM in a dashboard.

1 Access Grafana at `http://`*`visualization-vm-ip`*`:3000` by using the default user name **admin** and password `admin`.

2 Add Prometheus as the first data source, connecting to *visualization-vm-ip* on port 9090.

3 Create a dashboard with the vGPU metrics.

For more information on configuring a dashboard using a Prometheus data source, see the Grafana documentation.

Triton Inference Server

You can use a deep learning VM with a Triton Inference Server for loading a model repository and receive inference requests.

See the Triton Inference Server page.

Table 3-5. Triton Inference Server Container Image

<table>
<tr><th>Component</th><th>Description</th></tr>
<tr><td>Container image</td><td>

```
nvcr.io/nvidia/tritonserver:ngc_image_tag
```

For example:

```
nvcr.io/nvidia/tritonserver:23.10-py3
```

For information on the Triton Inference Server container images that are supported for deep learning VMs, see VMware Deep Learning VM Release Notes.</td></tr>
<tr><td>Required inputs</td><td>To deploy a Triton Inference Server workload, you must set the OVF properties for the deep learning virtual machine in the following way:

- Use one of the following properties that are specific for the Triton Inference Server image.
  - Cloud-init script. Encode it in base64 format.

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

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

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

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

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

    docker run -d --gpus all --rm -p 8000:8000 -p
8001:8001 -p 8002:8002 -v /home/vmware/model_repository:/models
```
</td></tr>
</table>

Table 3-5. Triton Inference Server Container Image (continued)

Component	Description
	(continued below)

```
$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[@]}"
```

Table 3-5. Triton Inference Server Container Image (continued)

Component	Description

```
      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"
    }
```

For example, for tritonserver:23.10-py3, provide the following script in base64 format:


```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBw
LnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9i
YXNoCiAgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0
cmFwICdlcnJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3Jr
bG9hZCInIEVSUgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiICJzb2NrczUiCgog
ICAgREVGQVVMVF9SRUdfVVJJPSJudmNyLmlvIgogICAgUkVHSVNUUllfVVJJX1BBVEg9
JChncmVwIHJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4g
J3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCgogICAgaWYgW1sgLXogIiRS
RUdJU1RSWV9VUklfUEFUSCIgXV07IHRoZW4KICAgICAgIyBJZiBSRUdJU1RSWV9VUklf
UEFUSCBpcyBudWxsIG9yIGVtcHR5LCB1c2UgdGhlIGRlZmF1bHQgdmFsdWUKICAgICAg
UkVHSVNUUllfVVJJX1BBVEg9JERFRkFVTFRfUkVHX1VSSQogICAgICBlY2hvICJSRUdJ
U1RSWV9VUklfUEFUSCB3YXMgZW1wdHkuIFVzaW5nIGRlZmF1bHQ6ICRSRUdJU1RSWV9V
UklfUEFUSCIKICAgIGZpCiAgICAKICAgICMgSWYgUkVHSVNUUllfVVJJX1BBVEggY29u
dGFpbnMgJy8nLCBleHRyYWN0IHRoZSBVUkkgcGFydAogICAgaWYgW1sgJFJFR0lTVFJZ
X1VSSV9QQVRIID09ICoiLyIqIF1dOyB0aGVuCiAgICAgIFJFR0lTVFJZX1VSST0kKGVj
aG8gIiRSRUdJU1RSWV9VUklfUEFUSCIgfCBjdXQgLWQnLycgLWYxKQogICAgZWxzZQog
ICAgICBSRUdJU1RSWV9VUkk9JFJFR0lTVFJZX1VSSV9QQVRICiAgICBmaQogIAogICAg
UkVHSVNUUllfVVNFUk5BTUU9JChncmVwIHJlZ2lzdHJ5LXVzZXIgL29wdC9kbHZtL292
ZillbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3An
KQogICAgUkVHSVNUUllfUEFTU1dPUkQ9JChncmVwIHJlZ2lzdHJ5LXBhc3N3ZCAvb3B0
L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwp
LiovXDEvcCcpCiAgICBpZiBbWyAtbiAiJFJFR0lTVFJZX1VTRVJOQU1FIiAmJiAtbiAi
JFJFR0lTVFJZX1BBU1NXT1JEIiBdXTsgdGhlbgogICAgICBkb2NrZXIgbG9naW4gLXUg
JFJFR0lTVFJZX1VTRVJOQU1FIC1wICRSRUdJU1RSWV9QQVNTV09SRCAkUkVHSVNUUllf
VVJJCiAgICBlbHNlCiAgICAgIGVjaG8gIldhcm5pbmc6IHRoZSByZWdpc3RyeSdzIHVz
ZXJuYW1lIGFuZCBwYXNzd29yZCBhcmUgaW52YWxpZCwgU2tpcHBpbmcgRG9ja2VyIGxv
Z2luLiIKICAgIGZpCgogICAgZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tcm0gLXAg
ODAwMDo4MDAwIC1wIDgwMDE6ODAwMSAtcCA4MDAyOjgwMDIgLXYgL2hvbWUvdm13YXJl
L21vZGVsX3JlcG9zaXRvcnk6L21vZGVscyAkUkVHSVNUUllfVVJJX1BBVEgvbnZpZGlh
L3RyaXRvbnNlcnZlcjoyMy4xMC1weTMgdHJpdG9uc2VydmVyIC0tbW9kZWwtcmVwb3Np
dG9yeT0vbW9kZWxzIC0tbW9kZWwtY29udHJvbC1tb2RlPXBvbGwKCi0gcGF0aDogL29w
dC9kbHZtL3V0aWxzLnNoCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwK
```

Table 3-5. Triton Inference Server Container Image (continued)

Component	Description


```
ICAgICMhL2Jpbi9iYXNoCiAgICBlcnJvcl9leGl0KCkgewogICAgICBlY2hvICJFcnJv
cjogJDEiID4mMgogICAgICB2bXRvb2xzZCAtLWNtZCAiaW5mby1zZXQgZ3Vlc3RpbmZv
LnZtc2VydmljZS5ib290c3RyYXAuY29uZGl0aW9uIGZhbHNlLCBETFdvcmtsb2FkRmFp
bHVyZSwgJDEiCiAgICAgIGV4aXQgMQogICAgfQoKICAgIGNoZWNrX3Byb3RvY29sKCkg
ewogICAgICBsb2NhbCBwcm94eV91cmw9JDEKICAgICAgc2hpZnQKICAgICAgbG9jYWwg
c3VwcG9ydGVkX3Byb3RvY29scz0oIiRAIikKICAgICAgaWYgW1sgLW4gIiR7cHJveHlf
dXJsfSIgXV07IHRoZW4KICAgICAgICBsb2NhbCBwcm90b2NvbD0kKGVjaG8gIiR7cHJv
eHlfdXJsfSIgfCBhd2sgLUYgJzovLycgJ3tpZiAoTkYgPiAxKSBwcmludCAkMTsgZWxz
ZSBwcmludCAiIn0nKQogICAgICAgIGlmIFsgLXogIiRwcm90b2NvbCIgXTsgdGhlbgog
ICAgICAgICAgZWNobyAiTm8gc3BlY2lmaWMgcHJvdG9jb2wgcHJvdmlkZWQuIFNraXBw
aW5nIHByb3RvY29sIGNoZWNrLiIKICAgICAgICAgIHJldHVybiAwCiAgICAgICAgZmkK
ICAgICAgICBsb2NhbCBwcm90b2NvbF9pbmNsdWRlZD1mYWxzZQogICAgICAgIGZvciB2
YXIgaW4gIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iOyBkbwogICAgICAgICAgaWYg
W1sgIiR7cHJvdG9jb2x9IiA9PSAiJHt2YXJ9IiBdXTsgdGhlbgogICAgICAgICAgICBw
cm90b2NvbF9pbmNsdWRlZD10cnVlCiAgICAgICAgICAgIGJyZWFrCiAgICAgICAgICBm
aQogICAgICAgIGRvbmUKICAgICAgICBpZiBbWyAiJHtwcm90b2NvbF9pbmNsdWRlZH0i
ID09IGZhbHNlIF1dOyB0aGVuCiAgICAgICAgICBlcnJvcl9leGl0ICJVbnN1cHBvcnRl
ZCBwcm90b2NvbDogJHtwcm90b2NvbH0uIFN1cHBvcnRlZCBwcm90b2NvbHMgYXJlOiAk
e3N1cHBvcnRlZF9wcm90b2NvbHNbKl19IgogICAgICAgIGZpCiAgICAgIGZpCiAgICB9
CgogICAgIyAkQDogbGlzdCBvZiBzdXBwb3J0ZWQgcHJvdG9jb2xzCiAgICBzZXRfcHJv
eHkoKSB7CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIpCgogICAg
ICBDT05GSUdfSlNPTl9CQVNFNjQ9JChncmVwICdjb25maWctanNvbicgL29wdC9kbHZt
L292Zi1lbnYueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wx
L3AnKQogICAgICBDT05GSUdfSlNPTj0kKGVjaG8gJHtDT05GSUdfSlNPTl9CQVNFNjR9
IHwgYmFzZTY0IC0tZGVjb2RlKQoKICAgICAgSFRUUF9QUk9YWV9VUkw9JChlY2hvICIk
e0NPTkZJR19KU09OfSIgfCBqcSAtciAnLmh0dHBfcHJveHkgLy8gZW1wdHknKQogICAg
ICBIVFRQU19QUk9YWV9VUkw9JChlY2hvICIke0NPTkZJR19KU09OfSIgfCBqcSAtciAn
Lmh0dHBzX3Byb3h5IC8vIGVtcHR5JykKICAgICAgaWYgW1sgJD8gLW5lIDAgfHwgKC16
ICIke0hUVFBfUFJPWFlfVVJMfSIgJiYgLXogIiR7SFRUUFNfUFJPWFlfVVJMfSIpIF1d
OyB0aGVuCiAgICAgICAgZWNobyAiSW5mbzogVGhlIGNvbmZpZy1qc29uIHdhcyBwYXJz
ZWQsIGJ1dCBubyBwcm94eSBzZXR0aW5ncyB3ZXJlIGZvdW5kLiIKICAgICAgICByZXR1
cm4gMAogICAgICBmaQoKICAgICAgY2hlY2tfcHJvdG9jb2wgIiR7SFRUUF9QUk9YWV9V
Ukx9IiAiJHtzdXBwb3J0ZWRfcHJvdG9jb2xzW0BdfSIKICAgICAgY2hlY2tfcHJvdG9j
b2wgIiR7SFRUUFNfUFJPWFlfVVJMfSIgIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0i
CgogICAgICBpZiAhIGdyZXAgLXEgJ2h0dHBfcHJveHknIC9ldGMvZW52aXJvbm1lbnQ7
IHRoZW4KICAgICAgICBlY2hvICJleHBvcnQgaHR0cF9wcm94eT0ke0hUVFBfUFJPWFlf
VVJMfQogICAgICAgIGV4cG9ydCBodHRwc19wcm94eT0ke0hUVFBTX1BST1hZX1VSTH0K
ICAgICAgICBleHBvcnQgSFRUUF9QUk9YWT0ke0hUVFBfUFJPWFlfVVJMfQogICAgICAg
IGV4cG9ydCBIVFRQU19QUk9YWT0ke0hUVFBTX1BST1hZX1VSTH0KICAgICAgICBleHBv
cnQgbm9fcHJveHk9bG9jYWxob3N0LDEyNy4wLjAuMSIgPj4gL2V0Yy9lbnZpcm9ubWVu
dAogICAgICAgIHNvdXJjZSAvZXRjL2Vudmlyb25tZW50CiAgICAgIGZpCiAgICAgIAog
ICAgICAjIENvbmZpZ3VyZSBEb2NrZXIgdG8gdXNlIGEgcHJveHkKICAgICAgbWtkaXIg
LXAgL2V0Yy9zeXN0ZW1kL3N5c3RlbS9kb2NrZXIuc2VydmljZS5kCiAgICAgIGVjaG8g
IltTZXJ2aWNlXQogICAgICBFbnZpcm9ubWVudD1cIkhUVFBfUFJPWFk9JHtIVFRQX1BS
T1hZX1VSTH1cIgogICAgICBFbnZpcm9ubWVudD1cIkhUVFBTX1BST1hZPSR7SFRUUFNf
UFJPWFlfVVJMfVwiCiAgICAgIEVudmlyb25tZW50PVwiTk9fUFJPWFk9bG9jYWxob3N0
LDEyNy4wLjAuMVwiIiA+IC9ldGMvc3lzdGVtZC9zeXN0ZW0vZG9ja2VyLnNlcnZpY2Uu
ZC9wcm94eS5jb25mCiAgICAgIHN5c3RlbWN0bCBkYWVtb24tcmVsb2FkCiAgICAgIHN5
c3RlbWN0bCByZXN0YXJ0IGRvY2tlcgoKICAgICAgZWNobyAiSW5mbzogZG9ja2VyIGFu
ZCBzeXN0ZW0gZW52aXJvbm1lbnQgYXJlIG5vdyBjb25maWd1cmVkIHRvIHVzZSB0aGUg
cHJveHkgc2V0dGluZ3MiCiAgICB9
```


which corresponds to the following script in plain-text format:

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

Table 3-5. Triton Inference Server Container Image (continued)

Component	Description

```
    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
```

Table 3-5. Triton Inference Server Container Image (continued)

Component	Description

```
          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
```

Table 3-5. Triton Inference Server Container Image (continued)

<table>
<tr><th>Component</th><th>Description</th></tr>
<tr><td></td><td>

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

- Image one-liner encoded in base64 format

```
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
```

For example, for tritonserver:23.10-py3, provide the following script in base 64 format:


```
ZG9ja2VyIHJ1biAtZCAtLWdwdXMgYWxsIC0tcm0gLXA4MDAwOjgwMDAgLXA4MDAxOjgw
MDEgLXA4MDAyOjgwMDIgLXYgL2hvbWUvdm13YXJlL21vZGVsX3JlcG9zaXRvcnk6L21v
ZGVscyBudmNyLmlvL252aWRpYS90cml0b25zZXJ2ZXI6MjMuMTAtcHkzIHRyaXRvbnNl
cnZlciAtLW1vZGVsLXJlcG9zaXRvcnk9L21vZGVscyAtLW1vZGVsLWNvbnRyb2wtbW9k
ZT1wb2xs
```


which corresponds to the following script in plain-text format:

```
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
```

- Enter the vGPU guest driver installation properties, such as `vgpu-license` and `nvidia-portal-api-key`.
- Provide values for the properties required for a disconnected environment as needed.

See OVF Properties of Deep Learning VMs.
</td></tr>
<tr><td>Output</td><td>

- Installation logs for the vGPU guest driver in `/var/log/vgpu-install.log`.

  To verify that the vGPU guest driver is installed, log in to the VM over SSH and run the `nvidia-smi` command.
- Cloud-init script logs in `/var/log/dl.log`.
- Triton Inference Server container.

  To verify that the Triton Inference Server container is running, run the `sudo docker ps -a` and `sudo docker logs container_id` commands.

The model repository for the Triton Inference Server is in `/home/vmware/model_repository`. Initially, the model repository is empty and the initial log of the Triton Inference Server instance shows that no model is loaded.
</td></tr>
</table>

Create a Model Repository

To load your model for model inference, perform these steps:

1 Create the model repository for your model.

See the NVIDIA Triton Inference Server Model Repository documentation .

2 Copy the model repository to `/home/vmware/model_repository` so that the Triton Inference Server can load it.

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

Send Model Inference Requests

1 Verify that the Triton Inference Server is healthy and models are ready by running this command in the deep learning VM console.

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

2 Send a request to the model by running this command on the deep learning VM.

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

For more information on using the Triton Inference Server, see NVIDIA Triton Inference Server Model Repository documentation.

NVIDIA RAG

You can use a deep learning VM to build Retrieval Augmented Generation (RAG) solutions with an Llama2 model.

See the NVIDIA RAG Applications Docker Compose documentation (requires specific account permissions).

Table 3-6. NVIDIA RAG Container Image

<table>
<tr><th>Component</th><th>Description</th></tr>
<tr><td>Container images and models</td><td>

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

in the NVIDIA sample RAG pipeline.

For information on the NVIDIA RAG container applications that are supported for deep learning VMs, see VMware Deep Learning VM Release Notes.</td></tr>
<tr><td>Required inputs</td><td>To deploy an NVIDIA RAG workload, you must set the OVF properties for the deep learning virtual machine in the following way:

- Enter a cloud-init script. Encode it in base64 format.

  For example, for version 24.03 of NVIDIA RAG, provide the following script:


```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBwLnN
oCiAgcGVybWlzc2lvbnM6ICcwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCi
AgICBzZXQgLWV1CiAgICBzb3VyY2UgL29wdC9kbHZtL3V0aWxzLnNoCiAgICB0cmFwICdlc
nJvcl9leGl0ICJVbmV4cGVjdGVkIGVycm9yIG9jY3VycyBhdCBkbCB3b3JrbG9hZCInIEVS
UgogICAgc2V0X3Byb3h5ICJodHRwIiAiaHR0cHMiCgogICAgY2F0IDw8RU9GID4gL29wdC9
kbHZtL2NvbmZpZy5qc29uCiAgICB7CiAgICAgICJfY29tbWVudCI6ICJUaGlzIHByb3ZpZG
VzIGRlZmF1bHQgc3VwcG9ydCBmb3IgUkFHOiBUZW5zb3JSVCBpbmZlcmVuY2UsIGxsYW1hM
i0xM2IgbW9kZWwsIGFuZCBIMTAweDIgR1BVIiwKICAgICAgInJhZyI6IHsKICAgICAgICAi
b3JnX25hbWUiOiAiY29jZndnYThqcTJjIiwKICAgICAgICAib3JnX3RlYW1fbmFtZSI6ICJ
uby10ZWFtIiwKICAgICAgICAicmFnX3JlcG9fbmFtZSI6ICJudmlkaWEvcGFpZiIsCiAgIC
AgICAgImxsbV9yZXBvX25hbWUiOiAibnZpZGlhL25pbSIsCiAgICAgICAgImVtYmVkX3Jlc
G9fbmFtZSI6ICJudmlkaWEvbmVtby1yZXRyaWV2ZXIiLAogICAgICAgICJyYWdfbmFtZSI6
ICJyYWctZG9ja2VyLWNvbXBvc2UiLAogICAgICAgICJyYWdfdmVyc2lvbiI6ICIyNC4wMyI
sCiAgICAgICAgImVtYmVkX25hbWUiOiAibnYtZW1iZWQtcWEiLAogICAgICAgICJlbWJlZF
90eXBlIjogIk5WLUVtYmVkLVFBIiwKICAgICAgICAiZW1iZWRfdmVyc2lvbiI6ICI0IiwKI
CAgICAgICAiaW5mZXJlbmNlX3R5cGUiOiAidHJ0IiwKICAgICAgICAibGxtX25hbWUiOiAi
bGxhbWEyLTEzYi1jaGF0IiwKICAgICAgICAibGxtX3ZlcnNpb24iOiAiaDEwMHgyX2ZwMTZ
fMjQuMDIiLAogICAgICAgICJudW1fZ3B1IjogIjIiLAogICAgICAgICJoZl90b2tlbiI6IC
JodWdnaW5nZmFjZSB0b2tlbiB0byBwdWxsIGxsbSBtb2RlbCwgdXBkYXRlIHdoZW4gdXNpb
mcgdmxsbSBpbmZlcmVuY2UiLAogICAgICAgICJoZl9yZXBvIjogImh1Z2dpbmdmYWNlIGxs
bSBtb2RlbCByZXBvc2l0b3J5LCB1cGRhdGUgd2hlbiB1c2luZyB2bGxtIGluZmVyZW5jZSI
KICAgICAgfQogICAgfQogICAgRU9GCiAgICBDT05GSUdfSlNPTj0kKGNhdCAiL29wdC9kbH
ZtL2NvbmZpZy5qc29uIikKICAgIElORkVSRU5DRV9UWVBFPSQoZWNobyAiJHtDT05GSUdfS
lNPTn0iIHwganEgLXIgJy5yYWcuaW5mZXJlbmNlX3R5cGUnKQogICAgaWYgWyAiJHtJTkZF
UkVOQ0VfVFlQRX0iID0gInRydCIgXTsgdGhlbgogICAgICByZXF1aXJlZF92YXJzPSgiT1J
HX05BTUUiICJPUkdfVEVBTV9OQU1FIiAiUkFHX1JFUE9fTkFNRSIgIkxMTV9SRVBPX05BTU
UiICJFTUJFRF9SRVBPX05BTUUiICJSQUdfTkFNRSIgIlJBR19WRVJTSU9OIiAiRU1CRURfT
kFNRSIgIkVNQkVEX1RZUEUiICJFTUJFRF9WRVJTSU9OIiAiTExNX05BTUUiICJMTE1fVkVS
U0lPTiIgIk5VTV9HUFUiKQogICAgZWxpZiBbICIke0lORkVSRU5DRV9UWVBFfSIgPSAidmx
sbSIgXTsgdGhlbgogICAgICByZXF1aXJlZF92YXJzPSgiT1JHX05BTUUiICJPUkdfVEVBTV
9OQU1FIiAiUkFHX1JFUE9fTkFNRSIgIkxMTV9SRVBPX05BTUUiICJFTUJFRF9SRVBPX05BT
UUiICJSQUdfTkFNRSIgIlJBR19WRVJTSU9OIiAiRU1CRURfTkFNRSIgIkVNQkVEX1RZUEUi
ICJFTUJFRF9WRVJTSU9OIiAiTExNX05BTUUiICJOVU1fR1BVIiAiSEZfVE9LRU4iICJIRl9
SRVBPIikKICAgIGVsc2UKICAgICAgZXJyb3JfZXhpdCAiSW5mZXJlbmNlIHR5cGUgJyR7SU
5GRVJFTkNFX1RZUEV9JyBpcyBub3QgcmVjb2duaXplZC4gTm8gYWN0aW9uIHdpbGwgYmUgd
GFrZW4uIgogICAgZmkKICAgIGZvciBpbmRleCBpbiAiJHshcmVxdWlyZWRfdmFyc1tAXX0i
OyBkbwogICAgICBrZXk9IiR7cmVxdWlyZWRfdmFyc1skaW5kZXhdfSIKICAgICAganFfcXV
lcnk9Ii5yYWcuJHtrZXksLH0gfCBzZWxlY3QgKC4hPW51bGwpIgogICAgICB2YWx1ZT0kKG
VjaG8gIiR7Q09ORklHX0pTT059IiB8IGpxIC1yICIke2pxX3F1ZXJ5fSIpCiAgICAgIGlmI
FtbIC16ICIke3ZhbHVlfSIgXV07IHRoZW4gCiAgICAgICAgZXJyb3JfZXhpdCAiJHtrZXl9
IGlzIHJlcXVpcmVkIGJ1dCBub3Qgc2V0LiIKICAgICAgZWxzZQogICAgICAgIGV2YWwgJHt
rZXl9PVwiIiR7dmFsdWV9IlwiCiAgICAgIGZpCiAgICBkb25lCgogICAgUkFHX1VSST0iJH
tSQUdfUkVQT19OQU1FfS8ke1JBR19OQU1FfToke1JBR19WRVJTSU9OfSIKICAgIEVNQkVEX
01PREVMX1VSST0iJHtFTUJFRF9SRVBPX05BTUV9LyR7RU1CRURfTkFNRX06JHtFTUJFRF9W
RVJTSU9OfSIKCiAgICBOR0NfQ0xJX1ZFUlNJT049IjMuNDEuMiIKICAgIE5HQ19DTElfVVJ
MPSJodHRwczovL2FwaS5uZ2MubnZpZGlhLmNvbS92Mi9yZXNvdXJjZXMvbnZpZGlhL25nYy
1hcHBzL25nY19jbGkvdmVyc2lvbnMvJHtOR0NfQ0xJX1ZFUlNJT059L2ZpbGVzL25nY2Nsa
```

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

Table 3-6. NVIDIA RAG Container Image (continued)

Component	Description
	V9saW51eC56aXAiCgogICAgbWtkaXIgLXAgL29wdC9kYXRhCiAgICBjZCAvb3B0L2RhdGEK CiAgICBpZiBbICEgLWYgLmZpbGVfZG93bmxvYWRlZCBdOyB0aGVuCiAgICAgICMgY2xlYW4 gdXAKICAgICAgcm0gLXJmIGNvbXBvc2UuZW52ICR7UkFHX05BTUV9KiAke0xMTV9OQU1FfS ogbmdjKiAke0VNQkVEX05BTUV9KiAqLmpzb24gLmZpbGVfZG93bmxvYWRlZAoKICAgICAgI yBpbnN0YWxsIG5nYy1jbGkKICAgICAgd2dldCAtLWNvbnRlbnQtZGlzcG9zaXRpb24gJHtO R0NfQ0xJX1VSTH0gLU8gbmdjY2xpX2xpbnV4LnppcCAmJiB1bnppcCBuZ2NjbGlfbGludXg uemlwCiAgICAgIGV4cG9ydCBQQVRIPWBwd2RgL25nYy1jbGk6JHtQQVRIfQoKICAgICAgQV BJS0VZPSIiCiAgICAgIFJFR19VUkk9Im52Y3IuaW8iCgogICAgICBpZiBbWyAiJChncmVwI HJlZ2lzdHJ5LXVyaSAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZWQgLW4gJ3MvLipvZTp2 YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpIiA9PSAqIiR7UkVHX1VSSX0iKiBdXTsgdGhlbgo gICAgICAgIEFQSUtFWT0kKGdyZXAgcmVnaXN0cnktcGFzc3dkIC9vcHQvZGx2bS9vdmYtZW 52LnhtbCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKICAgI CAgZmkKCiAgICAgIGlmIFsgLXogIiR7QVBJS0VZfSIgXTsgdGhlbgogICAgICAgICAgZXJy b3JfZXhpdCAiTm8gQVBJS0VZIGZvdW5kIgogICAgICBmaQoKICAgICAgIyBjb25maWcgbmd jLWNsaQogICAgICBta2RpciAtcCB+Ly5uZ2MKCiAgICAgIGNhdCA8PCBFT0YgPiB+Ly5uZ2 MvY29uZmlnCiAgICAgIFtDVVJSRU5UXQogICAgICBhcGlrZXkgPSAke0FQSUtFWX0KICAgI CAgZm9ybWF0X3R5cGUgPSBhc2NpaQogICAgICBvcmcgPSAke09SR19OQU1FfQogICAgICB0 ZWFtID0gJHtPUkdfVEVBTV9OQU1FfQogICAgICBhY2UgPSBuby1hY2UKICAgIEVPRgoKICA gICAgIyBuZ2MgZG9ja2VyIGxvZ2luCiAgICAgIGRvY2tlciBsb2dpbiBudmNyLmlvIC11IF wkb2F1dGh0b2tlbiAtcCAke0FQSUtFWX0KCiAgICAgICMgZG9ja2VyaHViIGxvZ2luIGZvc iBnZW5lcmFsIGNvbXBvbmVudHMsIGUuZy4gbWluaW8KICAgICAgRE9DS0VSSFVCX1VSST0k KGdyZXAgcmVnaXN0cnktMi11cmkgL29wdC9kbHZtL292ZiiilbnYueG1sIHwgc2VkIC1uICd zLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQogICAgICBET0NLRVJIVUJfVVNFUk 5BTUU9JChncmVwIHJlZ2lzdHJ5LTItdXNlciAvb3B0L2Rsdm0vb3ZmLWVudi54bWwgfCBzZ WQgLW4gJ3MvLipvZTp2YWx1ZT0iXChbXiJdKlwpLiovXDEvcCcpCiAgICAgIERPQ0tFUkhV Ql9QQVNTV09SRD0kKGdyZXAgcmVnaXN0cnktMi1wYXNzd2QgL29wdC9kbHZtL292ZiilbnY ueG1sIHwgc2VkIC1uICdzLy4qb2U6dmFsdWU9IlwoW14iXSpcKS4qL1wxL3AnKQoKICAgIC AgaWYgW1sgLW4gIiR7RE9DS0VSSFVCX1VTRVJOQU1FfSIgJiYgLW4gIiR7RE9DS0VSSFVCX 1BBU1NXT1JEfSIgXV07IHRoZW4KICAgICAgICBkb2NrZXIgbG9naW4gLXUgJHtET0NLRVJI VUJfVVNFUk5BTUV9IC1wICR7RE9DS0VSSFVCX1BBU1NXT1JEfQogICAgICBlbHNlCiAgICA gICAgZWNobyAiV2FybmluZzogRG9ja2VySHViIG5vdCBsb2dpbiIKICAgICAgZmkKCiAgIC AgICMgZ2V0IFJBRyBmaWxlcwogICAgICBuZ2MgcmVnaXN0cnkgcmVzb3VyY2UgZG93bmxvY WQtdmVyc2lvbiAke1JBR19VUkl9CgogICAgICAjIGdldCBsbG0gbW9kZWwKICAgICAgaWYg WyAiJHtJTkZFUkVOQ0VfVFlQRX0iID0gInRydCIgXTsgdGhlbgogICAgICAgIExMTV9NT0R FTF9VUkk9IiR7TExNX1JFUE9fTkFNRX0vJHtMTE1fTkFNRX06JHtMTE1fVkVSU0lPTn0iCi AgICAgICAgbmdjIHJlZ2lzdHJ5IG1vZGVsIGRvd25sb2FkLXZlcnNpb24gJHtMTE1fTU9ER UxfVVJJfQogICAgICAgIGNobW9kIC1SIG8rclggJHtMTE1fTkFNRX1fdiR7TExNX1ZFUlNJ T059CiAgICAgICAgTExNX01PREVMX0ZPTERFUj0iL29wdC9kYXRhLyR7TExNX05BTUV9X3Y ke0xMTV9WRVJTSU9OfSIKICAgICAgZWxpZiBbICIke0lORkVSRU5DRV9UWVBFfSIgPSAidm xsbSIgXTsgdGhlbgogICAgICAgIHBpcCBpbnN0YWxsIGh1Z2dpbmdmYWNlX2h1YgogICAgI CAgIGh1Z2dpbmdmYWNlLWNsaSBsb2dpbiAtLXRva2VuICR7SEZfVE9LRU59CiAgICAgICAg aHVnZ2luZ2ZhY2UtY2xpIGRvd25sb2FkIC0tcmVzdW1lLWRvd25sb2FkICR7SEZfUkVQT30 vJHtMTE1fTkFNRX0gLS1sb2NhbC1kaXIgJHtMTE1fTkFNRX0gLS1sb2NhbC1kaXItdXNlLX N5bWxpbmtzIEZhbHNlCiAgICAgICAgTExNX01PREVMX0ZPTERFUj0iL29wdC9kYXRhLyR7T ExNX05BTUV9IgogICAgICAgIGNhdCA8PCBFT0YgPiAke0xMTV9NT0RFTF9GT0xERVJ9L21v ZGVsX2NvbmZpZy55YW1sIAogICAgICAgIGVuZ2luZToKICAgICAgICAgIG1vZGVsOiAvbW9 kZWwtc3RvcmUKICAgICAgICAgIGVuZm9yY2VfZWFnZXI6IGZhbHNlCiAgICAgICAgICBtYX hfY29udGV4dF9sZW5fdG9fY2FwdHVyZTogODE5MgogICAgICAgICAgbWF4X251bV9zZXFzO iAyNTYKICAgICAgICAgIGR0eXBlOiBmbG9hdDE2CiAgICAgICAgICB0ZW5zb3JfcGFyYWxs ZWxfc2l6ZTogJHtOVU1fR1BVfQogICAgICAgICAgZ3B1X21lbW9yeV91dGlsaXphdGlvbjo gMC44CiAgICBFT0YKICAgICAgICBjaG1vZCAtUiBvK3JYICR7TExNX01PREVMX0ZPTERFUn 0KICAgICAgICBweXRob24zIC1jICJpbXBvcnQgeWFtbCwganNvbiwgc3lzOyBwcmludChqc 29uLmR1bXBzKHlhbWwuc2FmZV9sb2FkKHN5cy5zdGRpbi5yZWFkKCkpKSkiIDwgIiR7UkFH X05BTUV9X3Yke1JBR19WRVJTSU9OfS9yYWctYXBwLXRleHQtY2hhdGJvdC55YW1sIj4gcmF nLWFwcC10ZXh0LWNoYXRib3QuanNvbgogICAgICAgIGpxICcuc2VydmljZXMuIm5lbW9sbG 0taW5mZXJlbmNlIi5pbWFnZSA9ICJudmNyLmlvL252aWRpYS9uaW0vbmltX2xsbToyNC4wM i1kYXkwIiB8CiAgICAgICAgICAgICAgIC5zZXJ2aWNlcy4ibmVtb2xsbS1pbmZlcmVuY2UiLmNv bW1hbmQgPSAibmltX3ZsbG0gLS1tb2RlbF9uYW1lICR7TU9ERUxfTkFNRX0gLS1tb2RlbF9 jb25maWcgL21vZGVsLXN0b3JlL21vZGVsX2NvbmZpZy55YW1sIiB8CiAgICAgICAgICAgIC 5zZXJ2aWNlcy4ibmVtb2xsbS1pbmZlcmVuY2UiLnBvcnRzICs9IFsiODAwMDo4MDAwIl0gf AogICAgICAgICAgICAgICAuc2VydmljZXMuIm5lbW9sbG0taW5mZXJlbmNlIi5leHBvc2UgKz0g

Table 3-6. NVIDIA RAG Container Image (continued)

Component	Description
	(continued below)


```
WyI4MDAwIl0nIHJhZy1hcHAtdGV4dC1jaGF0Ym90Lmpzb24gPiB0ZW1wLmpzb24gJiYgbXY
gdGVtcC5qc29uIHJhZy1hcHAtdGV4dC1jaGF0Ym90Lmpzb24KICAgICAgICBweXRob24zIC
1jICJpbXBvcnQgeWFtbCwganNvbiwgc3lzOyBwcmludCh5YW1sLnNhZmVfZHVtcChqc29uL
mxvYWQoc3lzLnN0ZGluKSwgZGVmYXVsdF9mbG93X3N0eWxlPUZhbHNlLCBzb3J0X2tleXM9
RmFsc2UpKSIgPCByYWctYXBwLXRleHQtY2hhdGJvdC5qc29uID4gIiR7UkFHX05BTUV9X3Y
ke1JBR19WRVJTSU9OfS9yYWctYXBwLXRleHQtY2hhdGJvdC55YW1sIgogICAgICBmaQoKIC
AgICAgIyBnZXQgZW1iZWRkaW5nIG1vZGVscwogICAgICBuZ2MgcmVnaXN0cnkgbW9kZWwgZ
G93bmxvYWQtdmVyc2lvbiAke0VNQkVEX01PREVMX1VSSX0KICAgICAgY2htb2QgLVIgbyty
WCAke0VNQkVEX05BTUV9X3Yke0VNQkVEX1ZFUlNJT059CgogICAgICAjIGNvbmZpZyBjb21
wb3NlLmVudgogICAgICBjYXQgPDwgRU9GID4gY29tcG9zZS5lbnYKICAgICAgZXhwb3J0IE
1PREVMX0RJUkVDVE9SWT0iJHtMTE1fTU9ERUxfRk9MREVSfSIKICAgICAgZXhwb3J0IE1PR
EVMX05BTUU9JHtMTE1fTkFNRX0KICAgICAgZXhwb3J0IE5VTV9HUFU9JHtOVU1fR1BVfQog
ICAgICBleHBvcnQgQVBQX0NPTkZJR19GSUxFPS9kZXYvbnVsbAogICAgICBleHBvcnQgRU1
CRURESU5HX01PREVMX0RJUkVDVE9SWT0iL29wdC9kYXRhLyR7RU1CRURfTkFNRX1fdiR7RU
1CRURfVkVSU0lPTn0iCiAgICAgIGV4cG9ydCBFTUJFRERJTkdfTU9ERUxfTkFNRT0ke0VNQ
kVEX1RZUEV9CiAgICAgIGV4cG9ydCBFTUJFRERJTkdfTU9ERUxfQ0tQVF9OQU1FPSIke0VN
QkVEX1RZUEV9LSR7RU1CRURfVkVSU0lPTn0ubmVtbyIKICAgIEVPRgoKICAgICAgdG91Y2g
gLmZpbGVfZG93bmxvYWRlZAogICAgZmkKCiAgICAjIHN0YXJ0IE5HQyBSQUcKICAgIGRvY2
tlciBjb21wb3NlIC1mICR7UkFHX05BTUV9X3Yke1JBR19WRVJTSU9OfS9kb2NrZXItY29tc
G9zZS12ZWN0b3JkYi55YW1sIHVwIC1kIHBndmVjdG9yCiAgICBzb3VyY2UgY29tcG9zZS5l
bnY7IGRvY2tlciBjb21wb3NlIC1mICR7UkFHX05BTUV9X3Yke1JBR19WRVJTSU9OfS9yYWc
tYXBwLXRleHQtY2hhdGJvdC55YW1sIHVwIC1kCgotIHBhdGg6IC9vcHQvZGx2bS91dGlscy
5zaAogIHBlcm1pc3Npb25zOiAnMDc1NScKICBjb250ZW50OiB8CiAgICAjIS9iaW4vYmFza
AogICAgZXJyb3JfZXhpdCgpIHsKICAgICAgZWNobyAiRXJyb3I6ICQxIiA+JjIKICAgICAg
dm10b29sc2QgLS1jbWQgImluZm8tc2V0IGd1ZXN0aW5mby52bXNlcnZpY2UuYm9vdHN0cmF
wLmNvbmRpdGlvbiBmYWxzZSwgRExXb3JrbG9hZEZhaWx1cmUsICQxIgogICAgICBleGl0ID
EKICAgIH0KCiAgICBjaGVja19wcm90b2NvbCgpIHsKICAgICAgbG9jYWwgcHJveHlfdXJsP
SQxCiAgICAgIHNoaWZ0CiAgICAgIGxvY2FsIHN1cHBvcnRlZF9wcm90b2NvbHM9KCIkQCIp
CiAgICAgIGlmIFtbIC1uICIke3Byb3h5X3VybH0iIF1dOyB0aGVuCiAgICAgICAgbG9jYWw
gcHJvdG9jb2w9JChlY2hvICIke3Byb3h5X3VybH0iIHwgYXdrIC1GICc6Ly8nICd7aWYgKE
5GID4gMSkgcHJpbnQgJDE7IGVsc2UgcHJpbnQgIiJ9JykKICAgICAgICBpZiBbIC16ICIkc
HJvdG9jb2wiIF07IHRoZW4KICAgICAgICAgIGVjaG8gIk5vIHNwZWNpZmljIHByb3RvY29s
IHByb3ZpZGVkLiBTa2lwcGluZyBwcm90b2NvbCBjaGVjay4iCiAgICAgICAgICByZXR1cm4
gMAogICAgICAgIGZpCiAgICAgICAgbG9jYWwgcHJvdG9jb2xfaW5jbHVkZWQ9ZmFsc2UKIC
AgICAgICBmb3IgdmFyIGluICIke3N1cHBvcnRlZF9wcm90b2NvbHNbQF19IjsgZG8KICAgI
CAgICAgIGlmIFtbICIke3Byb3RvY29sfSIgPT0gIiR7dmFyfSIgXV07IHRoZW4KICAgICAg
ICAgICAgcHJvdG9jb2xfaW5jbHVkZWQ9dHJ1ZQogICAgICAgICAgICBicmVhawogICAgICA
gICAgZmkKICAgICAgICBkb25lCiAgICAgICAgaWYgW1sgIiR7cHJvdG9jb2xfaW5jbHVkZW
R9IiA9PSBmYWxzZSBdXTsgdGhlbgogICAgICAgICAgZXJyb3JfZXhpdCAiVW5zdXBwb3J0Z
WQgcHJvdG9jb2w6ICR7cHJvdG9jb2x9LiBTdXBwb3J0ZWQgcHJvdG9jb2xzIGFyZTogJHtz
dXBwb3J0ZWRfcHJvdG9jb2xzWypdfSIKICAgICAgICBmaQogICAgICBmaQogICAgfQoKICA
gICMgJEA6IGxpc3Qgb2Ygc3VwcG9ydGVkIHByb3RvY29scwogICAgc2V0X3Byb3h5KCkgew
ogICAgICBsb2NhbCBzdXBwb3J0ZWRfcHJvdG9jb2xzPSgiJEAiKQoKICAgICAgQ09ORklHX
0pTT05fQkFTRTY0PSQoZ3JlcCAnY29uZmlnLWpzb24nIC9vcHQvZGx2bS9vdmYtZW52Lnht
bCB8IHNlZCAtbiAncy8uKm9lOnZhbHVlPSJcKFteIl0qXCkuKi9cMS9wJykKICAgICAgQ09
ORklHX0pTT049JChlY2hvICR7Q09ORklHX0pTT05fQkFTRTY0fSB8IGJhc2U2NCAtLWRlY2
9kZSkKCiAgICAgIEhUVFBfUFJPWFlfVVJMPSQoZWNobyAiJHtDT05GSUdfSlNPTn0iIHwga
nEgLXIgJy5odHRwX3Byb3h5IC8vIGVtcHR5JykKICAgICAgSFRUUFNfUFJPWFlfVVJMPSQo
ZWNobyAiJHtDT05GSUdfSlNPTn0iIHwganEgLXIgJy5odHRwc19wcm94eSAvLyBlbXB0eSc
pCiAgICAgIGlmIFtbICQ/
IC1uZSAwIHx8ICgteiAiJHtIVFRQX1BST1hZX1VSTH0iICYmIC16ICIke0hUVFBTX1BST1h
ZX1VSTH0iKSBdXTsgdGhlbgogICAgICAgIGVjaG8gIkluZm86IFRoZSBjb25maWctanNvbi
B3YXMgcGFyc2VkLCBidXQgbm8gcHJveHkgc2V0dGluZ3Mgd2VyZSBmb3VuZC4iCiAgICAgI
CAgcmV0dXJuIDAKICAgICAgZmkKCiAgICAgIGNoZWNrX3Byb3RvY29sICIke0hUVFBfUFJP
WFlfVVJMfSIgIiR7c3VwcG9ydGVkX3Byb3RvY29sc1tAXX0iCiAgICAgIGNoZWNrX3Byb3R
vY29sICIke0hUVFBTX1BST1hZX1VSTH0iICIke3N1cHBvcnRlZF9wcm90b2NvbHNbQF19Ig
oKICAgICAgaWYgISBncmVwIC1xICdodHRwX3Byb3h5JyAvZXRjL2Vudmlyb25tZW50OyB0a
GVuCiAgICAgICAgZWNobyAiZXhwb3J0IGh0dHBfcHJveHk9JHtIVFRQX1BST1hZX1VSTH0K
ICAgICAgICBleHBvcnQgaHR0cHNfcHJveHk9JHtIVFRQU19QUk9YWV9VUkx9CiAgICAgICA
gZXhwb3J0IEhUVFBfUFJPWFk9JHtIVFRQX1BST1hZX1VSTH0KICAgICAgICBleHBvcnQgSF
RUUFNfUFJPWFk9JHtIVFRQU19QUk9YWV9VUkx9CiAgICAgICAgZXhwb3J0IG5vX3Byb3h5P
```

Table 3-6. NVIDIA RAG Container Image (continued)

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

which corresponds to the following script in plain-text format:

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

    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
```

Table 3-6. NVIDIA RAG Container Image (continued)

Component	Description

```
      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}
```

Table 3-6. NVIDIA RAG Container Image (continued)

Component	Description
	(code continued below)

```
      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$
```

Table 3-6. NVIDIA RAG Container Image (continued)

Component	Description

```
{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
```

Table 3-6. NVIDIA RAG Container Image (continued)

<table>
<tr><th>Component</th><th>Description</th></tr>
<tr><td></td><td>

```
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"
    }
```


Enter the vGPU guest driver installation properties, such as `vgpu-license` and `nvidia-portal-api-key`.
Provide values for the properties required for a disconnected environment as needed.

See OVF Properties of Deep Learning VMs.</td></tr>
<tr><td>Output</td><td>
Installation logs for the vGPU guest driver in `/var/log/vgpu-install.log`.

To verify that the vGPU guest driver is installed, log in to the VM over SSH and run the `nvidia-smi` command.
Cloud-init script logs in `/var/log/dl.log`.
</td></tr>
</table>

Table 3-6. NVIDIA RAG Container Image (continued)

Component	Description
	To track deployment progress, run `tail -f /var/log/dl.log`. ■ Sample chatbot Web application that you can access at `http://dl_vm_ip:3001/orgs/nvidia/models/text-qa-chatbot` You can upload your own knowledge base.

Assign a Static IP Address to a Deep Learning VM in VMware Private AI Foundation with NVIDIA

By default, the deep learning VM images are configured with DHCP address assignment. If you want to deploy a deep learning VM with a static IP address directly on a vSphere cluster, you must add additional code to the cloud-init section.

On vSphere with Tanzu, IP address assignment is determined by the network configuration for the Supervisor in NSX.

Procedure

1 Create a cloud-init script in plain-text format for the DL workload you plan to use.

   See Deep Learning Workloads in VMware Private AI Foundation with NVIDIA.

2 Add the following additional code to the cloud-init script.

```
#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 Encode the resulting cloud-init script in base64 format.

4 Set the resulting cloud-init script in base64 format as a value to the `user-data` OVF parameter of the deep learning VM image.

Example: Assigning a Static IP Address to a CUDA Sample Workload

For an example deep learning VM with a Deep Learning Workloads in VMware Private AI Foundation with NVIDIA DL workload:

Deep Learning VM Element	Example Value
DL workload image	nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubi8
IP address	10.199.118.245
Subnet prefix	/25
Gateway	10.199.118.253
DNS servers	■ 10.142.7.1 ■ 10.132.7.1

you provide the following cloud-init code:


```
I2Nsb3VkLWNvbmZpZwp3cml0ZV9maWxlczoKLSBwYXRoOiAvb3B0L2Rsdm0vZGxfYXBwLnNoCiAgcGVybWlzc2lvbnM6IC
cwNzU1JwogIGNvbnRlbnQ6IHwKICAgICMhL2Jpbi9iYXNoCiAgICBkb2NrZXIgcnVuIC1kIG52Y3IuaW8vbnZpZGlhL2s4
cy9jdWRhLXNhbXBsZTp2ZWN0b3JhZGQtY3VkYTExLjcuMS11Ymk4CgptYW5hZ2VfZXRjX2hvc3RzOiB0cnVlCiAKd3JpdG
VfZmlsZXM6CiAgLSBwYXRoOiAvZXRjL25ldHBsYW4vNTAtY2xvdWQtaW5pdC55YW1sCiAgICBwZXJtaXNzaW9uczogJzA2
MDAnCiAgICBjb250ZW50OiB8CiAgICAgIG5ldHdvcms6CiAgICAgICAgdmVyc2lvbjogMgogICAgICAgIHJlbmRlcmVyOi
BuZXR3b3JrZAogICAgICAgIGV0aGVybmV0czoKICAgICAgICAgIGVuczMzOgogICAgICAgICAgICBkaGNwNDogZmFsc2Ug
IyBkaXNhYmxlIERIQ1A0CiAgICAgICAgICAgIGFkZHJlc3NlczogWzEwLjE5OS4xMTguMjQ1LzI1XSAgIyBTZXQgdGhlIH
N0YXRpYyBJUCBhZGRyZXNzIGFuZCBtYXNrCiAgICAgICAgICAgIHJvdXRlczoKICAgICAgICAgICAgICAgIC0gdG86IGRl
ZmF1bHQKICAgICAgICAgICAgICAgICAgdmlhOiAxMC4xOTkuMTE4LjI1MyAjIENvbmZpZ3VyZSBnYXRld2F5CiAgICAgIC
AgICAgIG5hbWVzZXJ2ZXJzOgogICAgICAgICAgICAgIGFkZHJlc3NlczogWzEwLjE0Mi43LjEsIDEwLjEzMi43LjFdICMg
UHJvdmlkZSB0aGUgRE5TIHNlcnZlciBhZGRyZXNzLiBTZXBhcmF0ZSBtdWxpdHBsZSBETlMgc2VydmVyIGFkZHJlc3Nlcy
B3aXRoIGNvbW1hcy4KIApydW5jbWQ6CiAgLSBuZXRwbGFuIGFwcGx5
```


which corresponds to the following script in plain-text format:

```
#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
```

Configure a Deep Learning VM with a Proxy Server

To connect your deep learning VM to the Internet in a disconnected environment where Internet access is over a proxy server, you must provide the proxy server details in the `config.json` file in the virtual machine.

Procedure

1 Create a JSON file with the properties for proxy server.

Proxy server that does not require authentication	`{ "http_proxy": "protocol://ip-address-or-fqdn:port", "https_proxy": "protocol://ip-address-or-fqdn:port" }`
Proxy server that requires authentication	`{ "http_proxy": "protocol://username:password@ip-address-or-fqdn:port", "https_proxy": "protocol://username:password@ip-address-or-fqdn:port" }`

where:

- *protocol* is the communication protocol used by the proxy server, such as `http` or `https`.
- *username* and *password* are the credentials for authentication to the proxy server. If the proxy server does not require authentication, skip these parameters.
- *ip-address-or-fqdn*: The IP address or host name of the proxy server.
- *port*: The port number on which the proxy server is listening for incoming requests.

2 Encode the resulting JSON code in base64 format.

3 When you deploy the the deep learning VM image, add the encoded value to the `config-json` OVF property.

Troubleshooting Deep Learning VM Deployment in VMware Private AI Foundation with NVIDIA

The troubleshooting information about deployment of deep learning VM in VMware Private AI Foundation with NVIDIA provides solutions to potential problems that you might encounter.

- DL Workload Automation Is Not Performed

  After you deploy a deep learning VM in VMware Private AI Foundation with NVIDIA, the specified DL workload is not running.

- Downloading a DL Workload Fails Because of Invalid Authentication Credentials

  After you deploy a deep learning VM in VMware Private AI Foundation with NVIDIA, downloading the specified DL workload on the virtual machine fails with error log messages indicating invalid authentication credentials.

- Downloading the NVIDIA vGPU Guest Driver Fails Because of a Missing Download Link

  After you deploy a deep learning VM, downloading the specified vGPU guest driver on the virtual machine fails with error log messages indicating a missing download link or resource.

- The NVIDIA vGPU Guest Driver Is Shown as Unlicensed

  After a deep learning VM is deployed in VMware Private AI Foundation with NVIDIA, the NVIDIA vGPU guest driver status is unlicensed.

DL Workload Automation Is Not Performed

After you deploy a deep learning VM in VMware Private AI Foundation with NVIDIA, the specified DL workload is not running.

Problem

You deploy a deep learning VM with a DL workload to be pre-installed at initial startup. After the deep learning VM is started, the DL workload is not carried out.

Cause

1 The base64-encoded `user-data` or values of other OVF parameters, such as `image-oneliner` or `config-json` are saved or decoded incorrectly in the `/opt/dlvm/dl_app.sh` file. As a result, the DL workload script is not run.

2 The vGPU driver installation failed, causing the cloud-init script passed in the `user-data` OVF parameter to not be run. The cloud-init script relies on the successful installation of the NVIDIA vGPU driver.

Solution

On the deep learning VM, verify whether the DL workload is installed on the virtual machine and apply a solution accordingly.

<table>
<tr><th>Availability of the DL Workload</th><th>Solution</th></tr>
<tr><td>The DL workload components are not created on the virtual machine.</td><td>

If you are using a cloud-init script as input to the <code>user-data</code> OVF parameter, verify the following values:

Check the script that is encoded and input as <code>user-data</code>.

Make sure that <code>#cloud-config</code> appears on the first line and is included in the base64 equivalent.
Check the <code>path</code> parameter.
Check the base64 encoded string and make sure that the <code>user-data</code> value is correctly saved in <code>/opt/dlvm/dl_app.sh</code>.

If you are using other OVF parameters, verify the following values:

<code>image-oneliner</code>. Check the base64 encoded string and make sure that the one-line command is correctly saved in <code>/opt/dlvm/dl_app.sh</code>.
<code>config-json</code>. Check the base64 encoded string and make sure that the Docker compose file and <code>config.json</code>, if provided, are correctly saved in <code>/root/docker-compose.yaml</code> and <code>/root/.docker/config.json</code>.

For information about the OVF parameters of the latest deep learning VM image, see OVF Properties of Deep Learning VMs.</td></tr>
<tr><td>The DL workload components are created but the workload is not running.</td><td>

Check the error messages in <code>/var/log/vgpu-install.log</code>.
If you are using a cloud-init script as input to the <code>user-data</code> OVF parameter, check if the NVIDIA vGPU driver is installed and is working correctly. The cloud-init script is not run if the NVIDIA vGPU driver installation is unsuccessful.
</td></tr>
</table>

Downloading a DL Workload Fails Because of Invalid Authentication Credentials

After you deploy a deep learning VM in VMware Private AI Foundation with NVIDIA, downloading the specified DL workload on the virtual machine fails with error log messages indicating invalid authentication credentials.

Problem

If you are installing a DL workload container image, such as Triton Inference Server, TensorFlow or Pytorch, the `/var/log/dl.log` file contains the following message:

```
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>
```

For NVIDIA RAG, the `/var/log/dl.log` file contains the following message:

```
Error: Invalid apikey
chmod: cannot access 'llama2-13b-chat_vh100x2_fp16_24.02': No such file or directory

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

Cause

The authentication to the nvcr.io container registry has failed. As a result, the DL workload image cannot be downloaded on the virtual machine.

Solution

- Verify the credentials for login to the nvcr.io registry passed as OVF parameters or to the catalog setup wizard for private AI in VMware Aria Automation.
  - Registry: nvcr.io
  - Registry user account: $oauthtoken
  - Registry password: *NGC portal API key*
- Verify that the NVIDIA NGC portal API key has the permissions to access the required resources and that the key has not expired.

Downloading the NVIDIA vGPU Guest Driver Fails Because of a Missing Download Link

After you deploy a deep learning VM, downloading the specified vGPU guest driver on the virtual machine fails with error log messages indicating a missing download link or resource.

Problem

The `/var/log/vgpu-install.log` file contains one of the following messages:

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

```
No downloads found via API
```

Cause

The API key from the NVIDIA Licensing Portal that you pass as a value to the `nvidia-portal-api-key` OVF property or to the catalog setup wizard for private AI in VMware Aria Automation is invalid, expired or incorrectly formatted.

Solution

- Verify that the API key is valid.
- Verify that the API key is correctly entered.

The API key typically follows the UUID version 4 format `xxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

The NVIDIA vGPU Guest Driver Is Shown as Unlicensed

After a deep learning VM is deployed in VMware Private AI Foundation with NVIDIA, the NVIDIA vGPU guest driver status is unlicensed.

Problem

The `/var/log/vgpu-install.log` file contains one of the following messages:

```
License Status: Unlicensed
```

```
Unlicensed (Restricted)
```

Cause

The NVIDIA vGPU client configuration token that you pass as a value to the `vgpu-license` OVF property or to the catalog setup wizard for private AI in VMware Aria Automation is invalid, expired, or incorrectly formatted.

Solution

- Verify the validity of the client configuration token.
- Verify that the vGPU license is correctly formatted and follows the JWT token format, which typically looks like `eyxxxx.eyxxxxx.xxxxx`.

  You can decode the JWT token at jwt.io to check the expiration date and node server URL.
- The vGPU license token also saved in `/etc/nvidia/ClientConfigToken/client_configuration_token.tok`.
- To troubleshoot the problem further, run this command to check for specific error messages related to the communication to the NVIDIA license server.

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

To apply a new token, following these steps:

1 Replace the content of the `/etc/nvidia/ClientConfigToken/client_configuration_token.tok` file with a new token, run the following command:

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

2 Restart the NVIDIA service.

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

3 Verify the license status of the NVIDIA vGPU guest driver.

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

Deploying AI Workloads on TKG Clusters in VMware Private AI Foundation with NVIDIA

4

As a DevOps engineer, you can deploy container AI workloads on Tanzu Kubernetes Grid (TKG) clusters whose worker nodes are accelerated with NVIDIA GPUs.

For information about the support of AI workloads on TKG clusters, see About Deploying AI/ML Workloads on TKGS Clusters.

Read the following topics next:

- Provision a GPU-Accelerated TKG Cluster by Using a Self-Service Catalog in VMware Private AI Foundation with NVIDIA
- Provision a GPU-Accelerated TKG Cluster by Using the `kubectl` Command in a Connected VMware Private AI Foundation with NVIDIA Environment
- Provision a GPU-Accelerated TKG Cluster by Using the `kubectl` Command in a Disconnected VMware Private AI Foundation with NVIDIA Environment

Provision a GPU-Accelerated TKG Cluster by Using a Self-Service Catalog in VMware Private AI Foundation with NVIDIA

In VMware Private AI Foundation with NVIDIA, as a DevOps engineer, you can provision a TKG cluster accelerated with NVIDIA GPUs from VMware Aria Automation by using an AI Kubernetes Cluster self-service catalog items in Automation Service Broker. Then, you can deploy AI container images from NVIDIA NGC on the cluster.

Prerequisites

Verify with your cloud administrator that VMware Private AI Foundation with NVIDIA is configured. See Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment.

Procedure

- In Automation Service Broker, deploy an AI Kubernetes Cluster catalog item on the Supervisor instance configured by the cloud administrator.
  - For a non-RAG Tanzu Grid Kubernetes cluster, use the **AI Kubernetes Cluster** catalog item. See Deploy a GPU-accelerated Tanzu Kubernetes Grid cluster.
  - For a RAG-based Tanzu Grid Kubernetes Grid cluster, use the **AI Kubernetes RAG Cluster** catalog item. See Deploy a GPU-accelerated Tanzu Kubernetes Grid RAG cluster.

What to do next

Run an AI container image. In a connected environment, use the NVIDIA NGC catalog. In a disconnected environment, use the Harbor Registry on the Supervisor.

For a RAG-based Tanzu Grid Kubernetes Grid cluster, deploy a pgvector PostgreSQL database in VMware Data Services Manager and install the RAG Sample Pipeline from NVIDIA. See Deploy a RAG Workload on a TKG Cluster.

Provision a GPU-Accelerated TKG Cluster by Using the `kubectl` Command in a Connected VMware Private AI Foundation with NVIDIA Environment

In VMware Private AI Foundation with NVIDIA, as a DevOps engineer, by using the Kubernetes API, you provision a TKG cluster that uses NVIDIA GPUs. Then, you can deploy container AI workloads from the NVIDIA NGC catalog.

You use `kubectl` to deploy the TKG cluster on the namespace configured by the cloud administrator.

Prerequisites

Verify with the cloud administrator that the following prerequisites are in place for the AI-ready infrastructure.

- VMware Private AI Foundation with NVIDIA is configured. See Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment.
- In a disconnected environment, a content library with Ubuntu TKr images is added to the vSphere namespace for AI workloads. See Configure a Content Library with Ubuntu TKr for a Disconnected VMware Private AI Foundation with NVIDIA Environment.

Procedure

1 Log in to the Supervisor control plane.

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

2 Provision a TKG cluster and install the NVIDIA GPU Operator and NVIDIA Network Operator on it.

   See Cluster Operator Workflow for Deploying AI/ML Workloads on TKGS Clusters.

What to do next

Deploy an AI container image from the NVIDIA NGC catalog.

Provision a GPU-Accelerated TKG Cluster by Using the `kubectl` Command in a Disconnected VMware Private AI Foundation with NVIDIA Environment

In VMware Private AI Foundation with NVIDIA, as a DevOps engineer, by using the Kubernetes API, you provision a TKG cluster that uses NVIDIA GPUs. In a disconnected environment, you must additionally set up a local Ubuntu package repository and use the Harbor Registry for the Supervisor.

Prerequisites

Verify with the cloud administrator that the following prerequisites are in place for the AI-ready infrastructure.

- VMware Private AI Foundation with NVIDIA is configured for a disconnected environment. See Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment.
- A machine that has access to the Supervisor endpoint and to the local Helm repository hosting the for the NVIDIA GPU Operator chart definitions.

Procedure

1 Provision a TKG cluster on the vSphere namespace configured by the cloud administrator.

   See Provision a TKGS Cluster for NVIDIA vGPU.

2 Install the NVIDIA GPU Operator.

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

3 Monitor the operation.

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

What to do next

Deploy an AI container image from the Harbor Registry to the Supervisor.

Deploying RAG Workloads in VMware Private AI Foundation with NVIDIA

5

A Retrieval-Augmented Generation (RAG) workload consists of an LLM and external knowledge base with latest data, stored in a vector database. In VMware Private AI Foundation with NVIDIA, you can configure a RAG workload to use embeddings from a vector database managed by VMware Data Services Manager.

Read the following topics next:

Deploy a Vector Database in VMware Private AI Foundation with NVIDIA

If you plan to use Retrieval-Augmented Generation (RAG) with VMware Private AI Foundation with NVIDIA, set up a PostgreSQL database with pgvector by using VMware Data Services Manager.

You can create the database manually or create a self-service catalog in VMware Aria Automation that can be used by DevOps engineers and developers.

Prerequisites

- Verify that VMware Private AI Foundation with NVIDIA is available for the VI workload domain. See Deploying VMware Private AI Foundation with NVIDIA.
- Verify with your cloud administrator that the prerequisites for creating a PostgreSQL database are in place. See Creating Databases.
- Install the `psql` command line utility from the PostgreSQL Web site.

Procedure

1 Deploy a PostgreSQL database in the VI workload domain and get the connection string for the database.

You can use one of the following workflows. If you are a data scientist, you can directly deploy a database from VMware Aria Automation. Otherwise, you request a database deployment from your DSM Administrator or DSM User.

Deployment Workflow	Required User Role	Description
Deploy and get the connection string of a PostgreSQL database from VMware Aria Automation	Data scientist or DevOps engineer	See Deploy a Vector Database by Using a Self-Service Catalog Item in VMware Aria Automation.
Deploy and get the connection string of a PostgreSQL database from the VMware Data Services Manager Console.	DSM Administrator or DSM User, or a cloud administrator assigned one of these roles	See Creating Databases and Connecting to a Database.
Deploy and get the connection string of a PostgreSQL database by using the `kubectl` command	DSM Administrator or DSM User, or a DevOps engineer assigned one of these roles	See Enabling Self-Service Consumption of VMware Data Services Manager.

The connection string of the deployed database has the following format.

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

2 Activate the pgvector extension on the database by using the `psql` command line utility.

a Connect to the database.

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

b Activate the pgvector extension.

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

What to do next

Integrate the database in your RAG workload. See Deploy a Deep Learning VM with a RAG Workload and Deploy a RAG Workload on a TKG Cluster.

Deploy a Vector Database by Using a Self-Service Catalog Item in VMware Aria Automation

In VMware Private AI Foundation with NVIDIA, as data scientist or a DevOps engineer, you can deploy a vector database from VMware Aria Automation by using a self-service catalog item in Automation Service Broker.

Procedure

1 Log in to VMware Aria Automation and, in Automation Service Broker, locate the catalog item for database deployment according to the information from your cloud administrator.

   By default, the catalog item is called **DSM DBaaS**.

2 In the catalog item card, click **Request** and enter the details for the new PostgreSQL database.

   For more information on the settings for the database, see Creating Databases.

3 Get the connection string of the deployed database.

   a In Automation Service Broker, click **Deployments > Deployments** .

   b Select the deployment entry for the database.

   c On the **Topology** tab, select the cloud template for the database deployment and from the **Actions** menu for the template, select **Get Connection String**.

Results

For more information on provisioning and performing operations on databases in VMware Data Services Manager from VMware Aria Automation, see the `readme.md` file in the `AriaAutomation_DataServicesManager` bundle .

Deploy a Deep Learning VM with a RAG Workload

You can deploy a deep learning VM with an NVIDIA RAG workload using a pgvector PostgreSQL database managed by VMware Data Services Manager.

For information about the NVIDIA RAG workload, see the NVIDIA RAG Applications Docker Compose documentation (requires specific account permissions).

Prerequisites

- Verify that VMware Private AI Foundation with NVIDIA is configured. See Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment.
- Deploy a Vector Database in VMware Private AI Foundation with NVIDIA.

Procedure

1 If, as a data scientist, you are deploying the deep learning VM by using a catalog item in VMware Aria Automation, you provide the details of the pgvector PostgreSQL database after you deploy the virtual machine.

   a Deploy a RAG workstation in VMware Aria Automation.

   b Navigate to **Consume > Deployments > Deployments** and locate the deep learning VM deployment.

   c In the **Workstation VM** section, save the details for SSH login to the virtual machine.

d Log in to the deep learning VM over SSH by using the credentials available in Automation Service Broker.

e Add the following pgvector variables to the `/opt/data/compose.env` file:

```
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 Restart the NVIDIA RAG multi-container application by running the following commands.

For example, for 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 If, as a DevOps engineer, you are deploying the deep learning VM for a data scientist directly on the vSphere cluster or by using the `kubectl` command, create a cloud-init script and deploy the deep learning VM.

a Create a cloud-init script for NVIDIA RAG and the pgvector PostgreSQL database you have created.

You can modify the initial version of the cloud-init script for NVIDIA RAG. For example, for NVIDIA RAG 24.03 and a pgvector PostgreSQL database with connection details `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 Encode the cloud-init script to base64 format.

  You use a base 64 encoding tool, such as https://decode64base.com/ to generate the encoded versio of your cloud-init script.

c Deploy the deep learning VM, passing the base64 value of the cloud-init script to the `user-data` input parameter.

  See Deploy a Deep Learning VM Directly on a vSphere Cluster in VMware Private AI Foundation with NVIDIA or Deploy a Deep Learning VM by Using the kubectl Command in VMware Private AI Foundation with NVIDIA.

Deploy a RAG Workload on a TKG Cluster

As a DevOps engineer, on a TKG cluster in a Supervisor, you can deploy a RAG workload based on the RAG Sample Pipeline from NVIDIA that uses a pgvector PostgreSQL database managed by VMware Data Services Manager.

Prerequisites

- Verify that VMware Private AI Foundation with NVIDIA is available for the VI workload domain. See Chapter 2 Preparing VMware Cloud Foundation for Private AI Workload Deployment.
- Deploy a Vector Database in VMware Private AI Foundation with NVIDIA.

Procedure

1 Provision a GPU-accelated TKG cluster.

  You can use one of the following workflows.

Provisioning Workflow	Steps
By using a catalog item in VMware Aria Automation	Deploy a GPU-accelerated Tanzu Kubernetes Grid RAG cluster.
By using the `kubectl` comamnd	1 Provision a GPU-Accelerated TKG Cluster by using the `kubectl` command. ■ For a connected environment, see Provision a GPU-Accelerated TKG Cluster by Using the kubectl Command in a Connected VMware Private AI Foundation with NVIDIA Environment . ■ For a disconnected environment, see Provision a GPU-Accelerated TKG Cluster by Using the kubectl Command in a Disconnected VMware Private AI Foundation with NVIDIA Environment. 2 Install the RAG LLM Operator. See Install the RAG LLM Operator.

2 If you used the `kubectl` command to provision the TKG cluster, install the NVIDIA RAG LLM Operator on the TKG cluster.

   See Install the RAG LLM Operator.

   During deployment, the **AI Kubernetes RAG Cluster** catalog item in VMware Aria Automation automatically installs the NVIDIA RAG LLM Operator on the TKG cluster.

3 Download the manifests for the NVIDIA sample RAG pipeline.

   See Sample RAG Pipeline.

4 Configure the sample RAG pipeline with the pgvector PostgreSQL database.

   a Edit the sample pipeline YAML file.

      See Step 4 in Sample RAG Pipeline.

   b In the YAML file, configure the sample pipeline with the pgvector PostgreSQL database by using the database's connection string.

      See Vector Database for RAG Sample Pipeline .

5 To provide an external IP for the sample chat application, in the YAML file, set `frontend.service.type` to `loadBalancer`.

6 Start the sample RAG pipeline.

   See Sample RAG Pipeline.

7 To access the sample chat application, run the following command to get the application's external IP address.

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

8 In a Web browser, open the sample chat application at **`http://`** ***`application_external_ip`*** **`:3001/orgs/nvidia/models/text-qa-chatbot`**.

Monitoring VMware Private AI Foundation with NVIDIA

6

You can monitor GPU metrics at the cluster and host level in the vSphere Client and VMware Aria Operations.

In VMware Aria Operations, you can monitor GPU metrics at the cluster, host system and host properties levels. For more information, see Private AI (GPU) Dashboards and Properties for vCenter Server Components in VMware Aria Operations.

In the vSphere Client, you can monitor GPU metrics in the following way:

- At the host level. See Hosts Performance Charts in vSphere.
- At the cluster level in custom charts. See Working with Advanced and Custom Charts in vSphere.