FlexPod for Accelerated RAG Pipeline with NVIDIA NIM and Cisco Webex

Design and Deployment Guide

Published: October 2024

In partnership with:

About the Cisco Validated Design Program

The Cisco Validated Design (CVD) program consists of systems and solutions designed, tested, and documented to facilitate faster, more reliable, and more predictable customer deployments. For more information, go to: http://www.cisco.com/go/designzone.

Executive Summary

The FlexPod Datacenter solution offers a validated design for implementing Cisco and NetApp technologies to create shared private and public cloud infrastructures. Through their partnership, Cisco and NetApp have developed a range of FlexPod solutions that support strategic data center platforms. The success of the FlexPod solution is attributed to its capacity to adapt and integrate technological and product advancements in management, computing, storage, and networking.

This document is the Cisco Validated Design and Deployment Guide for FlexPod for Accelerated RAG Pipeline with NVIDIA Inference Microservices (NIM) and Cisco Webex on the latest FlexPod Datacenter design with Cisco UCS X-Series based compute nodes, Cisco Intersight, NVIDIA GPUs, Red Hat OpenShift Container Platform (OCP) 4.16, Cisco Nexus switches, and NetApp ONTAP storage.

Generative Artificial Intelligence (Generative AI) is revolutionizing industries by driving innovations across various use cases. However, integrating generative AI into enterprise environments presents unique challenges. Establishing the right infrastructure with adequate computational resources is essential.

Retrieval Augmented Generation (RAG) is a cutting-edge technique in natural language processing (NLP) that is used to optimize the output of LLMs (Large Language Models) with dynamic domain specific data fetched from external sources. RAG is an end-to-end architecture that combines information retrieval component with response generator. Retrieval component fetches relevant document snippet in real time from corpus of knowledge. knowledge. Generators take the input query along with retrieval chunks and generate natural response for the users. This hybrid approach enhances the effectiveness of AI applications by minimizing inaccuracies and boosting the relevance and faithfulness of the generated content.

This solution implements a RAG pipeline using NVIDA NIM microservices deployed on the infrastructure. The RAG RAG pipeline is integrated with Cisco Webex Chat bot to run inferencing.

This document provides design and deployment guidance for running Retrieval Augmented Generation pipeline using NVIDIA NIM microservices deployed on FlexPod Datacenter and Cisco Webex Chat bot for inferencing.

For information about the FlexPod design and deployment details, including the configuration of various elements of the design and associated best practices, refer to Cisco Validated Designs for FlexPod, here: https://www.cisco.com/c/en/us/solutions/design-zone/data-center-design-guides/flexpod-design-guides.html .

Solution Overview

This chapter contains the following:

Introduction

The FlexPod Datacenter for Accelerated RAG Pipeline with NVIDIA NIM and Cisco Webex configuration represents a cohesive and flexible infrastructure solution that combines computing hardware, networking, storage, and software resources into a single, integrated architecture. Designed as a collaborative effort between Cisco, NetApp, NVIDIA, and Red Hat, this converged infrastructure platform is designed to deliver high levels of efficiency, scalability, and performance, suitable for Generative AI workloads. By standardizing on a validated design, organizations can accelerate deployment, reduce operational complexities, and confidently scale their IT operations to meet evolving business demands. This FlexPod architecture leverages Cisco's Unified Computing System (UCS) servers, Cisco Nexus networking, and NetApp's innovative storage systems combined with Red Hat's container management platform and NVIDIA GPUs and AI software, providing a robust foundation for both virtualized and non-virtualized environments.

Retrieval Augmented Generation, or RAG, is the process of augmenting large language models with domain specific information. During RAG, the pipeline applies the knowledge and patterns that the model acquired during its training phase and couples that with user-supplied data sources to provide more informed responses to user queries.

The solution demonstrates how enterprises can design and implement a technology stack leveraging Generative Pre-trained Transformer (GPT), enabling them to privately run and manage their preferred AI large language models (LLMs) and related applications. It emphasizes Retrieval Augmented Generation as the key use case. The hardware and software components are seamlessly integrated, allowing customers to deploy the solution swiftly and cost-effectively, while mitigating many of the risks involved in independently researching, designing, building, and deploying similar systems on their own.

Audience

The intended audience of this document includes IT decision makers such as CTOs and CIOs, IT architects, sales engineers, field consultants, professional services, IT managers, partner engineers, and customers. It is designed for those interested in the design, deployment, and life cycle management of generative AI systems and applications, as well as those looking to leverage an infrastructure that enhances IT efficiency and fosters innovation.

Purpose of this Document

This document provides design and deployment guidance for running Retrieval Augmented Generation pipeline using NVIDIA NIM microservices deployed on FlexPod Datacenter and Cisco Webex Chat bot for inferencing.

What's New in this Release?

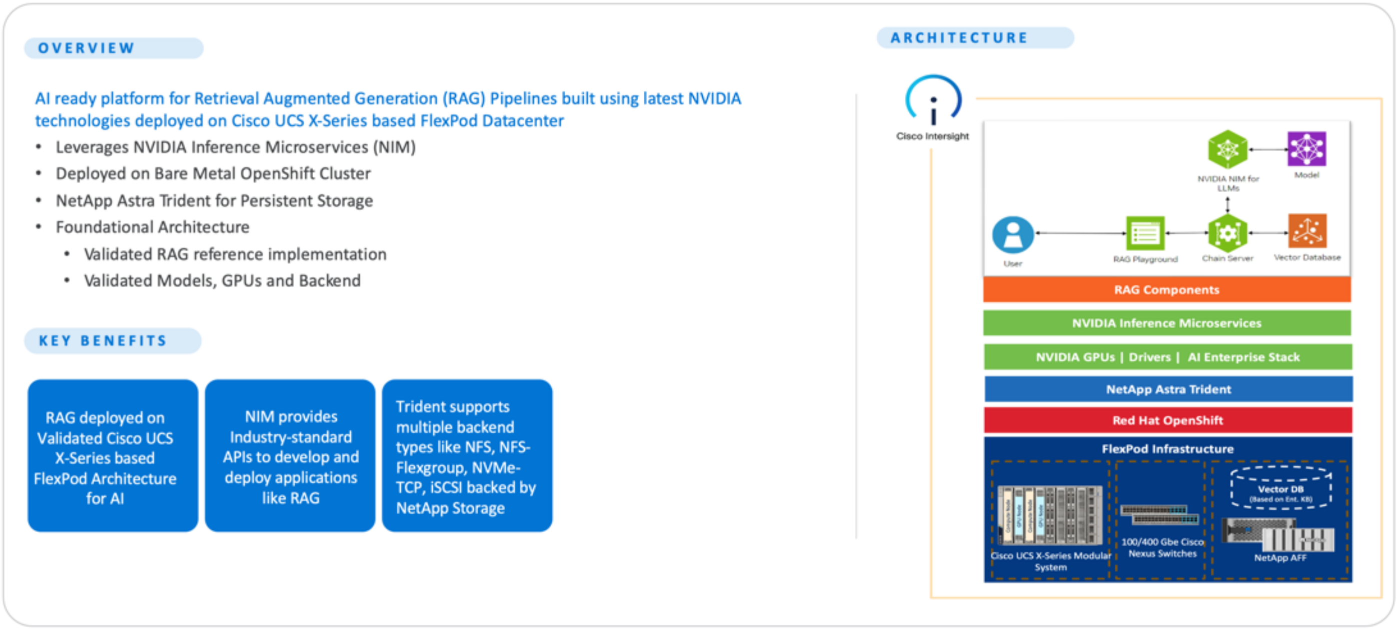
The following design elements are built on the CVD: [FlexPod Datacenter with Red Hat OCP Bare Metal Manual Configuration with Cisco UCS X-Series Direct Deployment Guide] to implement an end-to-end Retrieval Augmented Generation pipeline with NVIDIA NIM and Cisco Webex:

- Installation and configuration of RAG pipeline components on FlexPod Datacenter
    - NVIDIA GPUs
    - NVIDIA AI Enterprise (NVAIE)
    - NVIDIA NIM microservices
- Integrating RAG pipeline with Cisco Webex Chat bot
- RAG Evaluation
- NIM for LLM Benchmarking
- Milvus Benchmarking

Solution Summary

Figure 1 illustrates the solution summary.

Figure 1. Solution Overview



OVERVIEW

AI ready platform for Retrieval Augmented Generation (RAG) Pipelines built using latest NVIDIA technologies deployed on Cisco UCS X-Series based FlexPod Datacenter

- Leverages NVIDIA Inference Microservices (NIM)
- Deployed on Bare Metal OpenShift Cluster
- NetApp Astra Trident for Persistent Storage
- Foundational Architecture
    - Validated RAG reference implementation
    - Validated Models, GPUs and Backend

KEY BENEFITS

RAG deployed on Validated Cisco UCS X-Series based FlexPod Architecture for AI

NIM provides Industry-standard APIs to develop and deploy applications like RAG

Trident supports multiple backend types like NFS, NFS-Flexgroup, NVMe-TCP, iSCSI backed by NetApp Storage

The FlexPod Datacenter solution as a platform for Retrieval Augmented Generation offers the following key benefits:

- The ability to implement a Retrieval Augmented Generation pipeline quickly and easily on a powerful platform with high-speed persistent storage
- Evaluation of performance of platform components
- Simplified cloud-based management of solution components using policy-driven modular design
- Cooperative support model and Cisco Solution Support
- Easy to deploy, consume, and manage architecture, which saves time and resources required to research, procure, and integrate off-the-shelf components
- Support for component monitoring, solution automation and orchestration, and workload optimization
- Cisco Webex Chat bot integration with the RAG pipeline

Like all other FlexPod solution designs, FlexPod for Accelerated RAG Pipeline with NVIDIA NIM and Cisco Webex is configurable according to demand and usage. You can purchase exactly the infrastructure you need for your current application requirements and then scale-up/scale-out to meet future needs.

Technology Overview

This chapter contains the following:

- Retrieval Augmented Generation
- NVIDIA AI Enterprise
- NVIDIA Inference Microservices
- NVIDIA NIM for Large Language Models
- NVIDIA NIM for Text Embedding

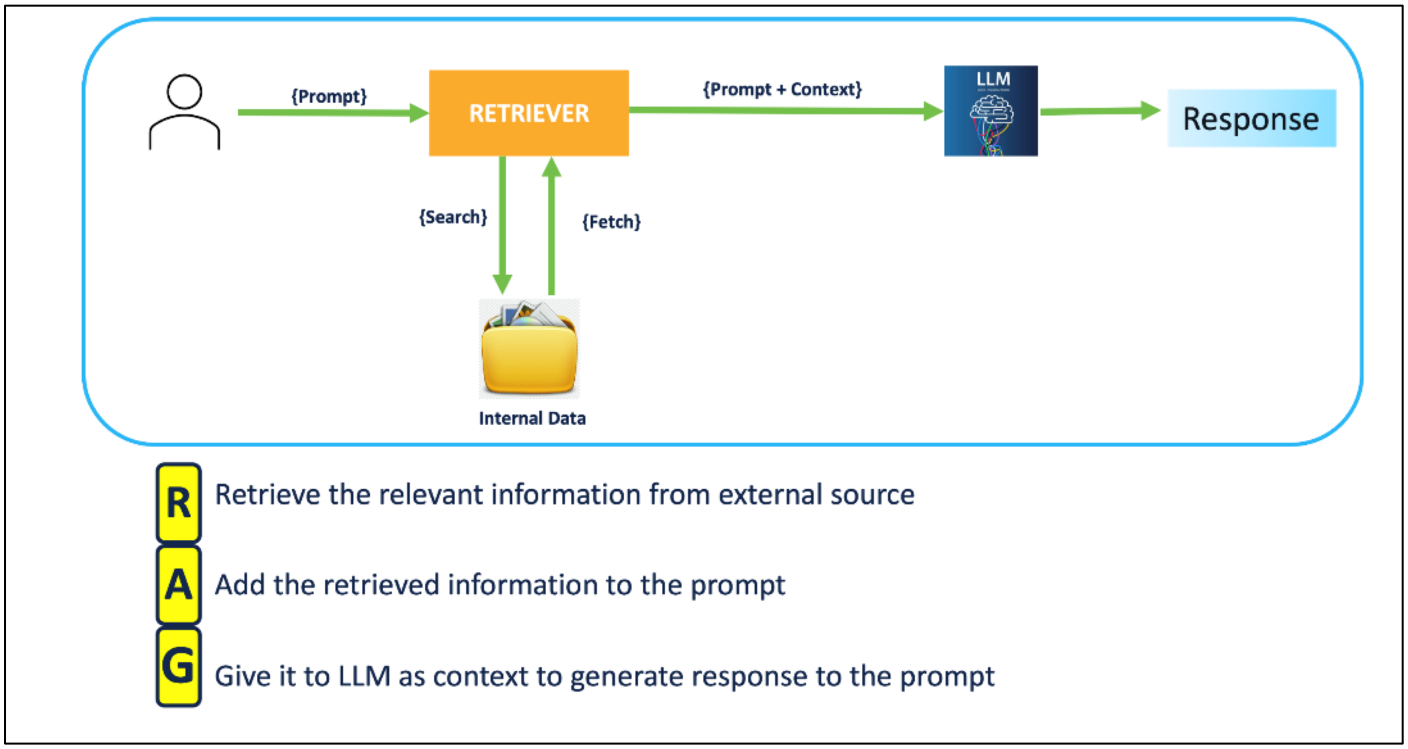
Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) is an enterprise application of Generative AI. RAG represents a category of large language model (LLM) applications that enhance the LLM's context by incorporating external data. It overcomes the limitation of knowledge cutoff date (events occurring after the model's training). LLMs lack access to an organization's internal data or services. This absence of up-to-date and domain-specific or organization-specific information prevents their effective use in enterprise applications.

RAG Pipeline

Figure 2 illustrates the RAG Pipeline overview.

Figure 2. RAG Pipeline overview

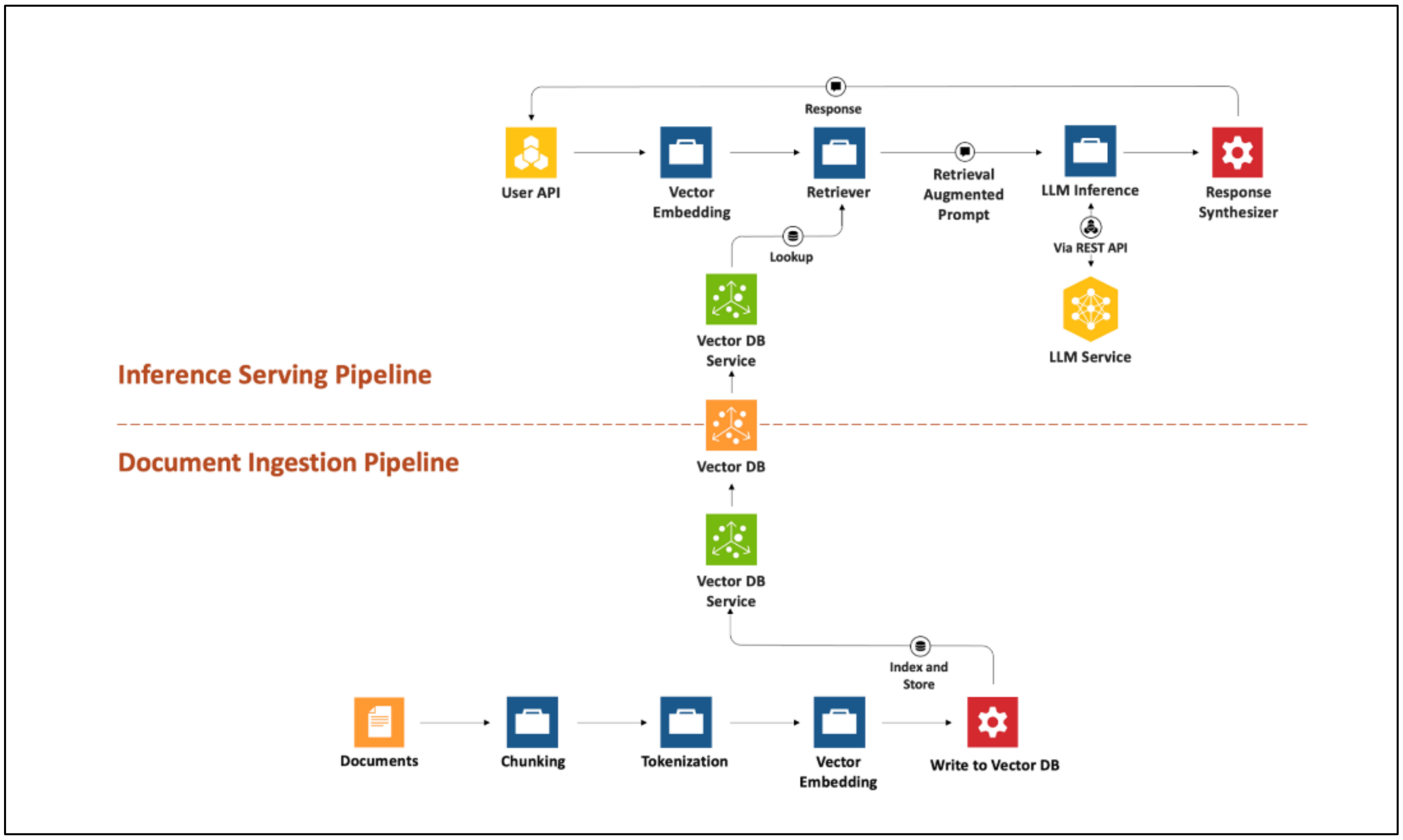

{Prompt}
RETRIEVER
{Prompt + Context}
LLM
Response
{Search}
{Fetch}
Internal Data
R
Retrieve the relevant information from external source
A
Add the retrieved information to the prompt
G
Give it to LLM as context to generate response to the prompt

In this pipeline, when you enter a prompt/query, document chunks relevant to the prompt are searched and fetched to the system. The retrieved relevant information is augmented to the prompt as context. LLM is asked to generate a response to the prompt in the context and the user receives the response.

RAG Architecture

RAG is an end-to-end architecture that combines information retrieval component with a response generator.

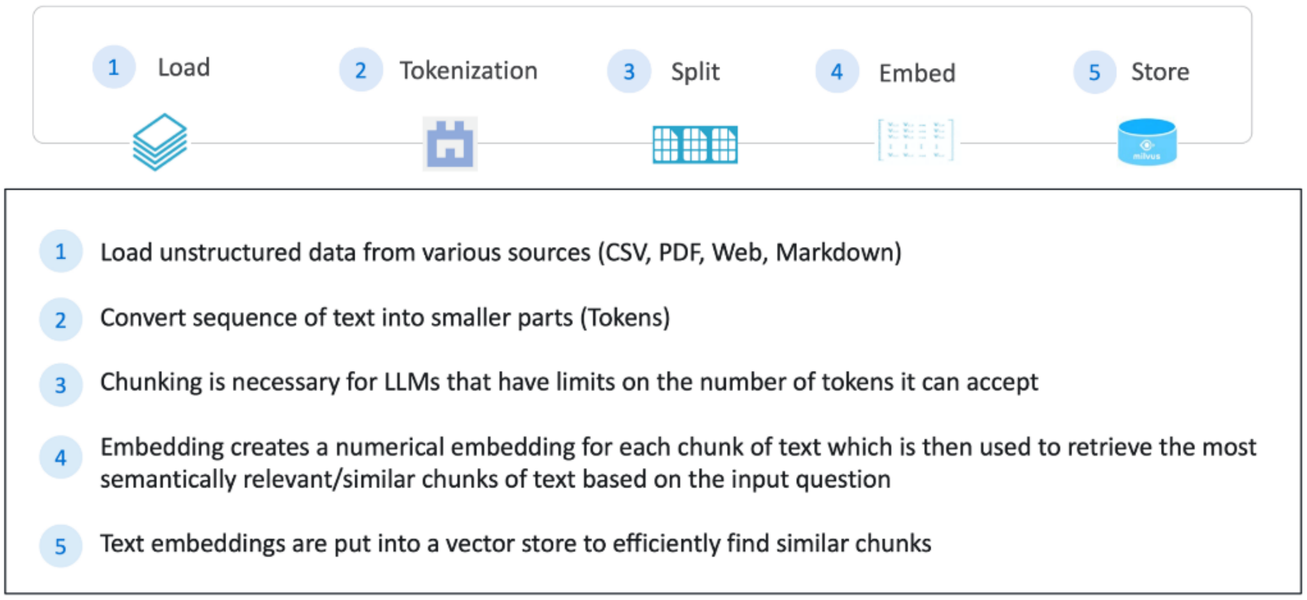
Figure 3. **RAG Architecture**



RAG can be broken into two process flows; document ingestion and inferencing.

Figure 4 illustrates the document ingestion pipeline.

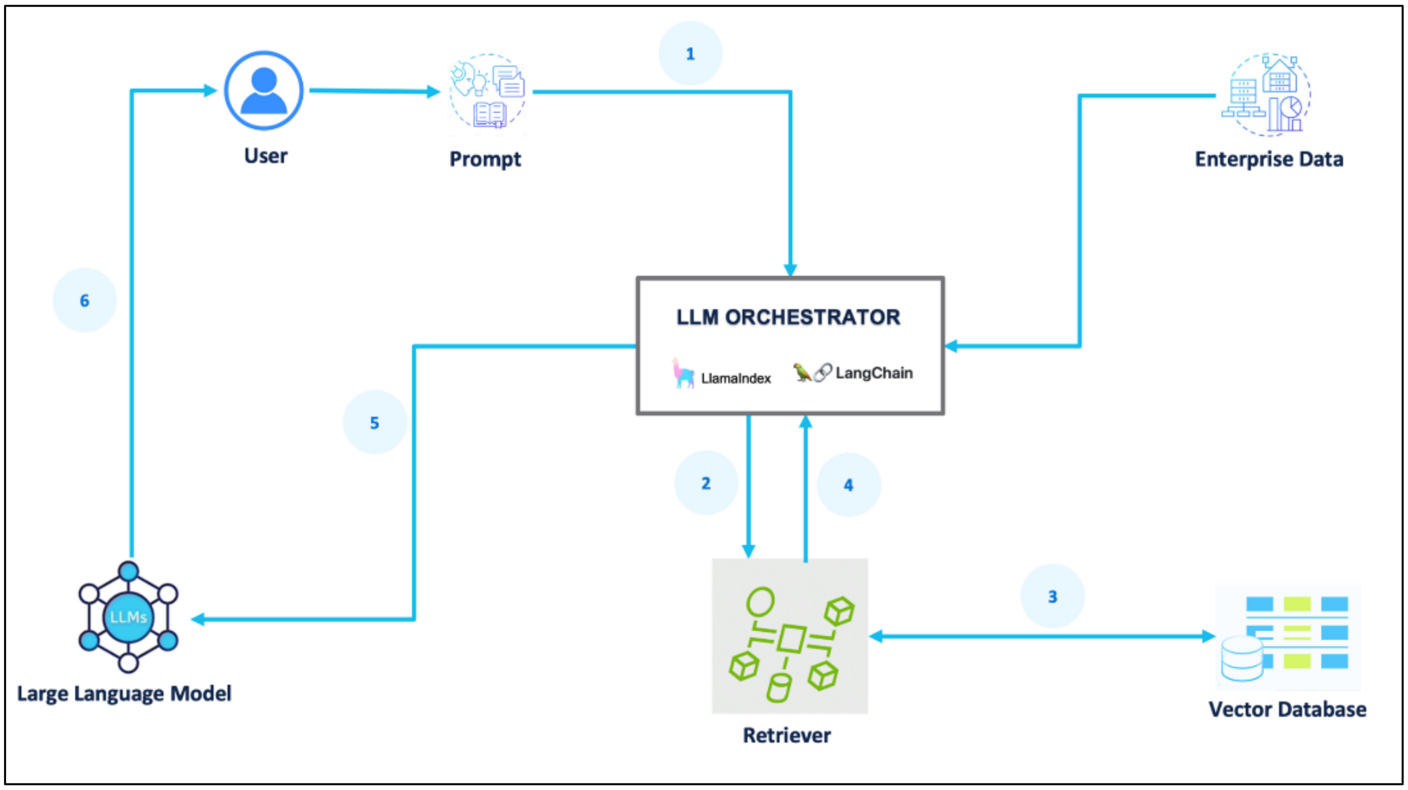
Figure 4. Document Ingestion Pipeline overview



1. Load unstructured data from various sources (CSV, PDF, Web, Markdown)
2. Convert sequence of text into smaller parts (Tokens)
3. Chunking is necessary for LLMs that have limits on the number of tokens it can accept
4. Embedding creates a numerical embedding for each chunk of text which is then used to retrieve the most semantically relevant/similar chunks of text based on the input question
5. Text embeddings are put into a vector store to efficiently find similar chunks

Figure 5 illustrates the inferencing pipeline.

Figure 5. Inferencing Pipeline overview



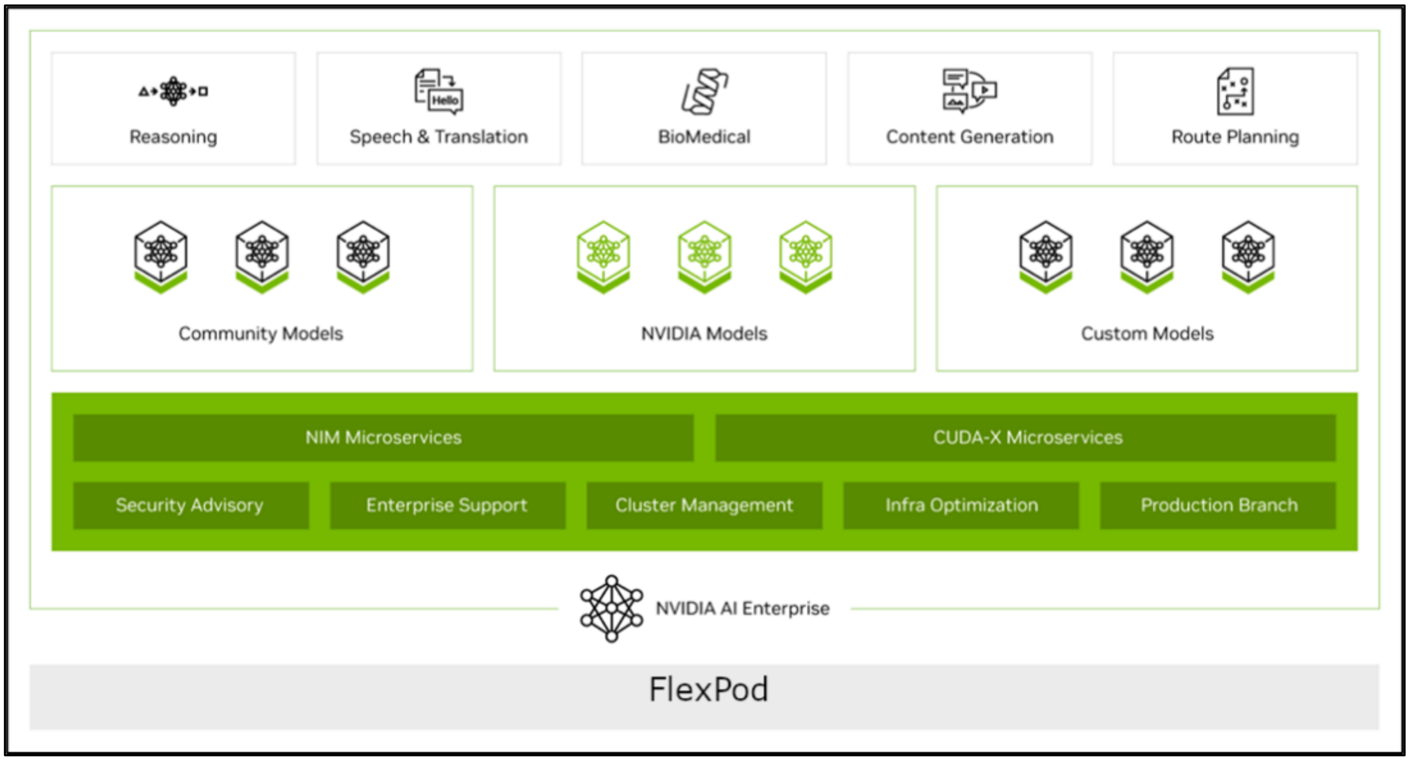
The process for the inference serving pipeline is as follows:

1. A prompt is passed to the LLM orchestrator.
2. The orchestrator sends a search query to the retriever.
3. The retriever fetches relevant information from the Vector Database.
4. The retriever returns the retrieved information to the orchestrator.
5. The orchestrator augments the original prompt with the context and sends it to the LLM.
6. The LLM responds with generated text/ response and presents it to the user.

NVIDIA AI Enterprise

The NVIDIA AI Enterprise (NVAIE) platform was deployed on Red Hat OpenShift as the foundation for the RAG pipeline. NVIDIA AI Enterprise simplifies the development and deployment of generative AI workloads, including Retrieval Augmented Generation, at scale.

Figure 6. NVIDIA AI Enterprise with FlexPod Datacenter

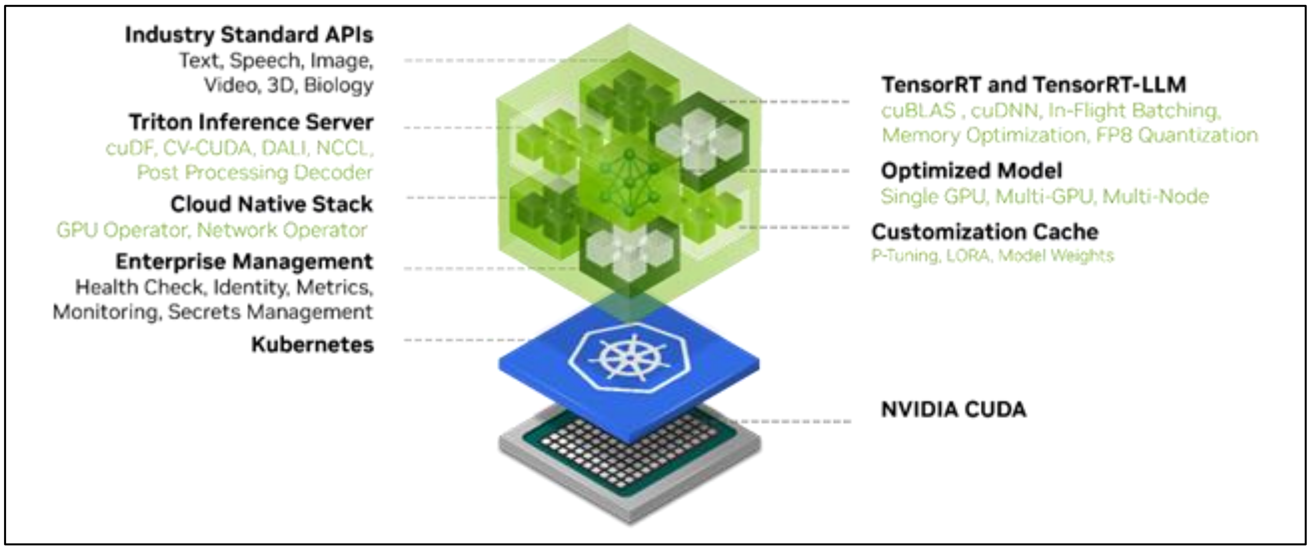


NVIDIA Inference Microservices

NVIDIA Inference Microservice (NIM), a component of NVIDIA AI Enterprise, offers an efficient route for creating AI-driven enterprise applications and deploying AI models in production environments. NIM consists of microservices that accelerate and simplify the deployment of generative AI models via automation using prebuilt containers, Helm charts, optimized models, and industry-standard APIs.

NIM simplifies the process for IT and DevOps teams to self-host large language models (LLMs) within their own managed environments. It provides developers with industry-standard APIs, enabling them to create applications such as copilots, chatbots, and AI assistants that can revolutionize their business operations. Content Generation, Sentiment Analysis, and Language Translation services are just a few additional examples of applications that can be rapidly deployed to meet various use cases. NIM ensures the quickest path to inference with unmatched performance.

Figure 7. **NVIDIA NIM for optimized AI inference**


Industry Standard APIs
Text, Speech, Image, Video, 3D, Biology
Triton Inference Server
cuDF, CV-CUDA, DALI, NCCL, Post Processing Decoder
Cloud Native Stack
GPU Operator, Network Operator
Enterprise Management
Health Check, Identity, Metrics, Monitoring, Secrets Management
Kubernetes
TensorRT and TensorRT-LLM
cuBLAS , cuDNN, In-Flight Batching, Memory Optimization, FP8 Quantization
Optimized Model
Single GPU, Multi-GPU, Multi-Node
Customization Cache
P-Tuning, LORA, Model Weights
NVIDIA CUDA

NIMs are distributed as container images tailored to specific models or model families. Each NIM is encapsulated in its own container and includes an optimized model. These containers come with a runtime compatible with any NVIDIA GPU that has adequate GPU memory, with certain model/GPU combinations being optimized for better performance. One or more GPUs can be passed through to containers via the NVIDIA Container Toolkit to provide the horsepower needed for any workload. NIM automatically retrieves the model from NGC (NVIDIA GPU Cloud), utilizing a local filesystem cache if available. Since all NIMs are constructed from a common base, once a NIM has been downloaded, acquiring additional NIMs becomes significantly faster. The NIM catalog currently offers nearly 150 models and agent blueprints.

Utilizing domain specific models, NIM caters to the demand for specialized solutions and enhanced performance through a range of pivotal features. It incorporates NVIDIA CUDA (Compute Unified Device Architecture) libraries and customized code designed for distinct fields like language, speech, video processing, healthcare, retail, and others. This method ensures that applications are precise and pertinent to their particular use cases. Think of it like a custom toolkit for each profession; just as a carpenter has specialized tools for woodworking, NIM provides tailored resources to meet the unique needs of various domains.

NIM is designed with a production-ready base container that offers a robust foundation for enterprise AI applications. It includes feature branches, thorough validation processes, enterprise support with service-level agreements (SLAs), and frequent security vulnerability updates. This optimized framework makes NIM an essential tool for deploying efficient, scalable, and tailored AI applications in production environments. Think of NIM as the bedrock of a skyscraper; just as a solid foundation is crucial for supporting the entire structure, NIM provides the necessary stability and resources for building scalable and reliable portable enterprise AI solutions.

NVIDIA NIM for Large Language Models

NVIDIA NIM for Large Language Models (LLMs) (NVIDIA NIM for LLMs) brings the power of state-of-the-art large language models (LLMs) to enterprise applications, providing unmatched natural language processing (NLP) and understanding capabilities.

Whether developing chatbots, content analyzers, or any application that needs to understand and generate human language – NVIDIA NIM for LLMs is the fastest path to inference. Built on the NVIDIA software platform, NVIDIA NIM brings state of the art GPU accelerated large language model serving.

High Performance Features

NVIDIA NIM for LLMs abstracts away model inference internals such as execution engine and runtime operations. NVIDIA NIM for LLMs provides the most performant option available whether it be with TensorRT, vLLM or LLM others.

- Scalable Deployment: NVIDIA NIM for LLMs is performant and can easily and seamlessly scale from a few users to millions.
- Advanced Language Models: Built on cutting-edge LLM architectures, NVIDIA NIM for LLMs provides optimized and pre-generated engines for a variety of popular models. NVIDIA NIM for LLMs includes tooling to help create GPU optimized models.
- Flexible Integration: Easily incorporate the microservice into existing workflows and applications. NVIDIA NIM for LLMs provides an OpenAI API compatible programming model and custom NVIDIA extensions for additional functionality.
- Enterprise-Grade Security: Data privacy is paramount. NVIDIA NIM for LLMs emphasizes security by using safetensors, constantly monitoring and patching CVEs in our stack and conducting internal penetration tests.

Applications

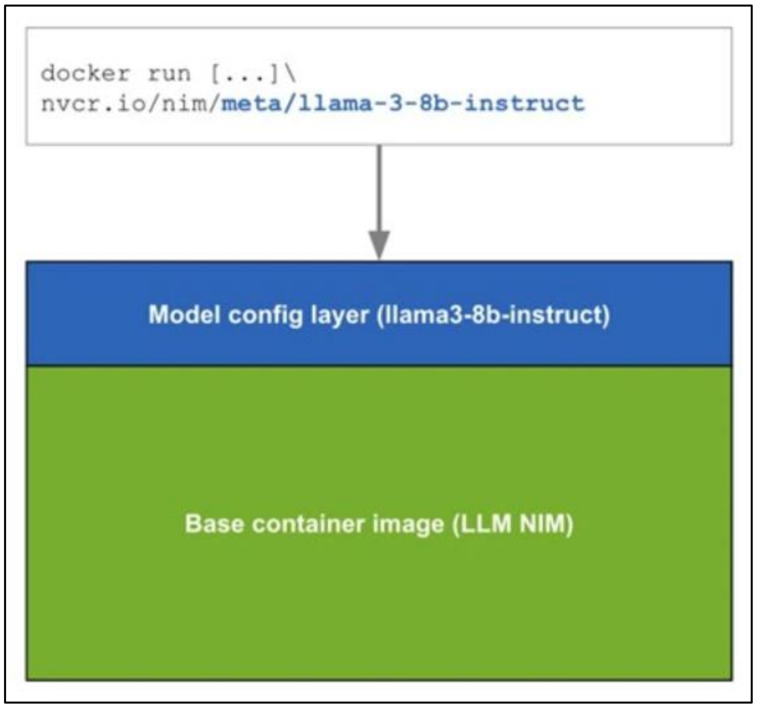
The potential applications of NVIDIA NIM for LLMs are vast, spanning across various industries and use cases:

- Chatbots & Virtual Assistants: Empower bots with human-like language understanding and responsive-ness.
- Content Generation & Summarization: Generate high-quality content or distill lengthy articles into concise summaries with ease.
- Sentiment Analysis: Understand user sentiments in real-time, driving better business decisions.
- Language Translation: Break language barriers with efficient and accurate translation services.

Architecture

NVIDIA NIM for LLMs is one of what will become many NIMs. Each NIM is its own Docker container with a model, such as `meta/llama3-8b-instruct`. These containers include the runtime capable of running the model on any NVIDIA GPU. The NIM automatically downloads the model from NGC, leveraging a local filesystem cache if available. Each NIM is built from a common base, so once a NIM has been downloaded, downloading additional NIMs is extremely fast.

Figure 8. NIM Architecture



```
docker run [...]\
nvcr.io/nim/meta/llama-3-8b-instruct
```

Model config layer (llama3-8b-instruct)
Base container image (LLM NIM)

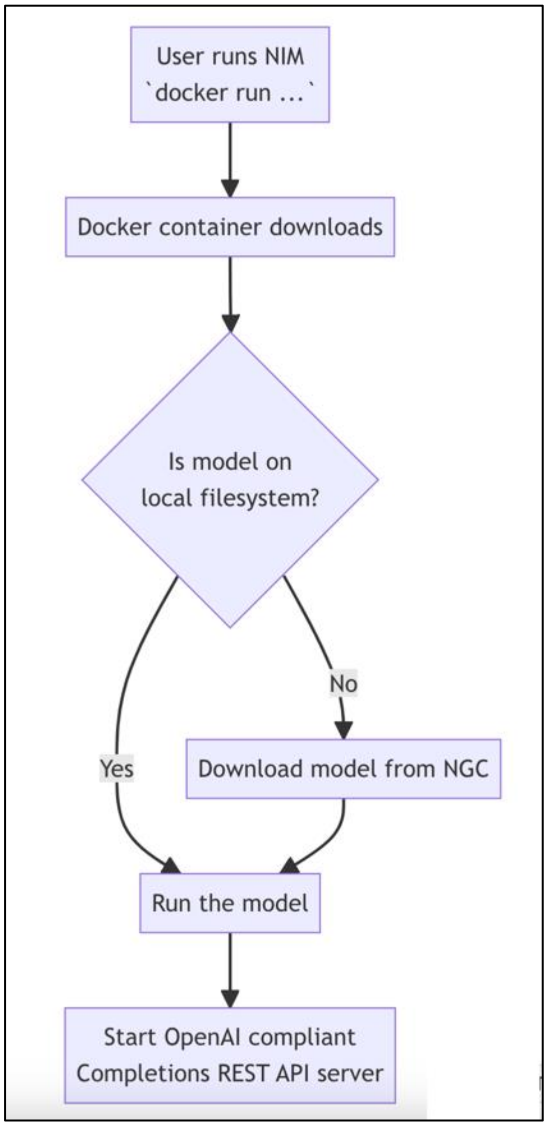
When a NIM is first deployed, NIM inspects the local hardware configuration, and the available optimized model in the model registry, and then automatically chooses the best version of the model for the available hardware. For a subset of NVIDIA GPUs, see: Support Matrix, NIM downloads the optimized TRT (TensorRT) engine and runs an inference using the TRT-LLM library. For all other NVIDIA GPUs, NIM downloads a non-optimized model and runs it using the vLLM library.

NIMs are distributed as NGC container images through the NVIDIA NGC Catalog. A security scan report is available for each container within the NGC catalog, which provides a security rating of that image, breakdown of CVE severity by package, and links to detailed information on CVEs.

Deployment Lifecycle

Figure 9 illustrates the deployment lifecycle.

Figure 9. **Deployment Lifecycle**

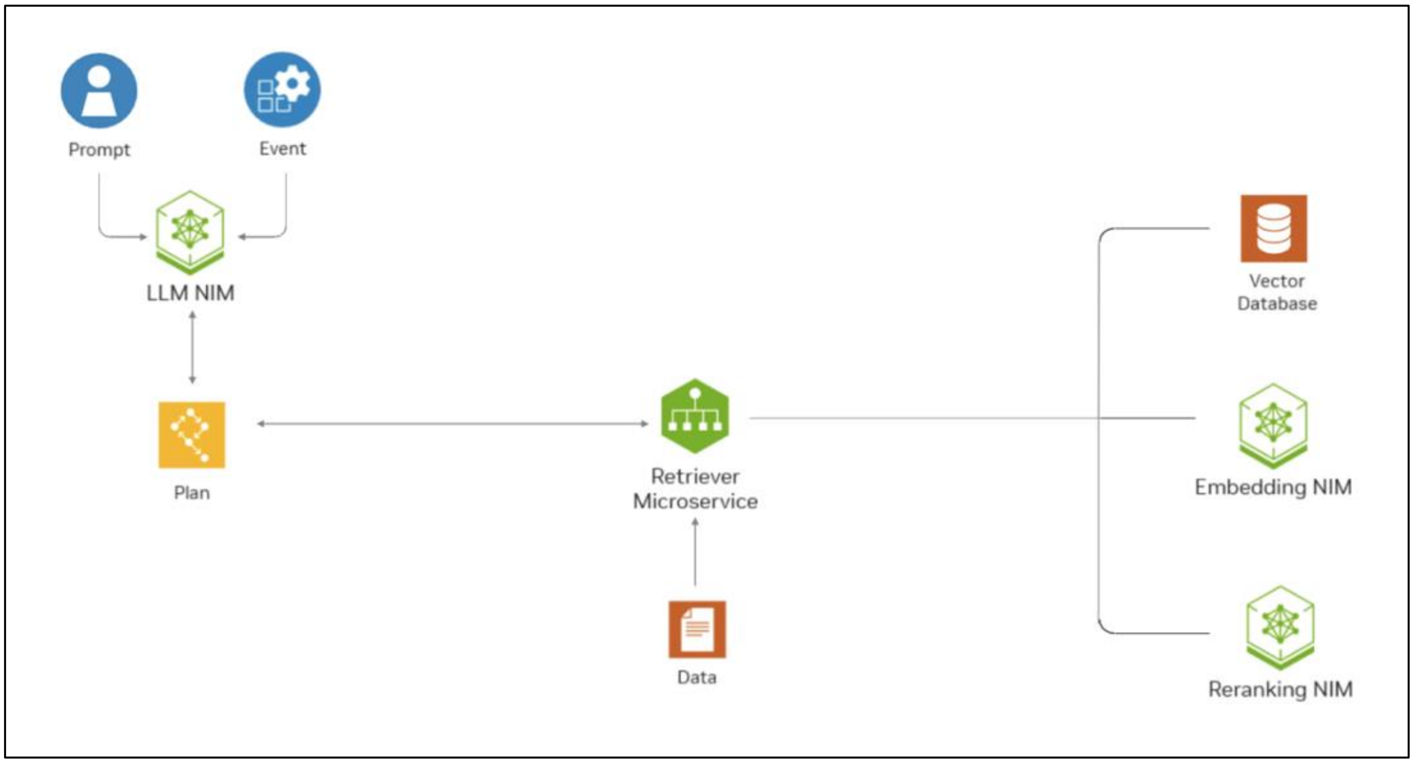

User runs NIM `docker run ...`
Docker container downloads
Is model on local filesystem?
No
Yes
Download model from NGC
Run the model
Start OpenAI compliant Completions REST API server

NVIDIA NIM for Text Embedding

NeMo Text Retriever NIM APIs facilitate access to optimized embedding models – essential components for RAG applications that deliver precise and faithful answers. By using NVIDIA software (including CUDA, TensorRT, and Triton Inference Server), the Text Retriever NIM provides the tools needed by developers to create ready-to-use, GPU-accelerated applications. NeMo Retriever Text Embedding NIM enhances the performance of text-based question-answering retrieval by generating optimized embeddings. For this RAG CVD, the Snowflake Arctic-Embed-L embedding model was harnessed to encode domain-specific content which was then stored in a vector database. The NIM combines that data with an embedded version of the user's query to deliver a relevant response.

Figure 10 shows how the Text Retriever NIM APIs can help a question-answering RAG application find the most relevant data in an enterprise setting.

Figure 10. **Text Retriever NIM APIs for RAG Application**



Enterprise-Ready Features

Text Embedding NIM comes with enterprise-ready features, such as a high-performance inference server, flexible integration, and enterprise-grade security.

- High Performance: Text Embedding NIM is optimized for high-performance deep learning inference with NVIDIA TensorRT and NVIDIA Triton Inference Server.
- Scalable Deployment: Text Embedding NIM seamlessly scales from a few users to millions.
- Flexible Integration: Text Embedding NIM can be easily incorporated into existing data pipelines and applications. Developers are provided with an OpenAI-compatible API in addition to custom NVIDIA extensions.
- Enterprise-Grade Security: Text Embedding NIM comes with security features such as the use of safetensors, continuous patching of CVEs, and constant monitoring with our internal penetration tests.

Solution Design

This chapter contains the following:

Design Requirements

This solution meets the following general design requirements:

- Resilient design across all layers of the infrastructure with no single point of failure
- Scalable and flexible design to add compute capacity, storage, or network bandwidth as needed supported by various models of each component
- Modular design that can be replicated to expand and grow as the needs of the business grow
- Simplified design with ability to integrate and automate with external automation tools
- Cloud-enabled design which can be configured, managed, and orchestrated from the cloud using GUI or APIs
- Repeatable design for accelerating the provisioning of end-to-end Retrieval Augmented Generation pipeline
- Provide a testing methodology to evaluate the performance of the solution
- Provide an example implementation of Cisco Webex Chat Bot integration with RAG

FlexPod Topology

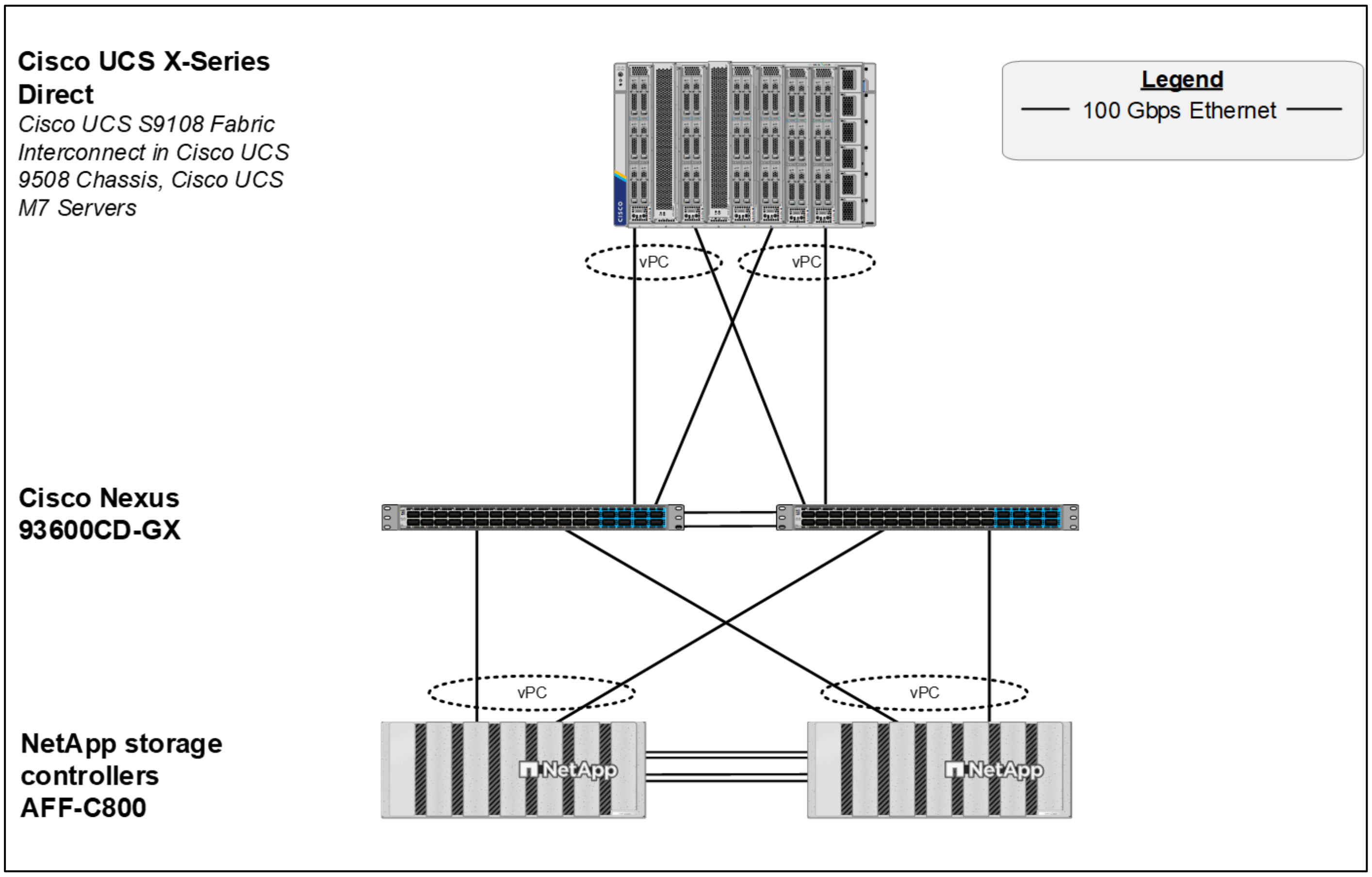
The FlexPod Datacenter for Accelerated RAG Pipeline with NVIDIA NIM and Cisco Webex is built using the following reference hardware components:

- 2 Cisco Nexus 93600CD-GX Switches in Cisco NX-OS mode provide the switching fabric. Other Cisco Nexus Switches are also supported.
- 2 Cisco UCS S9108 Fabric Interconnects (FIs) in the chassis provide the chassis connectivity. At least 2 100 Gigabit Ethernet ports from each FI, configured as a Port-Channel, are connected to each Cisco Nexus 93600CD-GX switch. 25 Gigabit Ethernet connectivity is also supported as well as other versions of the Cisco UCS FI that would be used with Intelligent Fabric Modules (IFMs) in the chassis.
- 1 Cisco UCS X9508 Chassis contains 6 Cisco UCS X210C M7 servers and 2 Cisco UCS X440p PCIe Nodes each with 2 NVIDIA L40S GPUs. Other configurations of servers with and without GPUs are also supported.

- 1 NetApp AFF C800 HA pair connects to the Cisco Nexus 93600CD-GX Switches using two 100 GE ports from each controller configured as a Port-Channel. 25 Gigabit Ethernet connectivity is also supported as well as other NetApp AFF, ASA, and FAS storage controllers.

Figure 11 shows various hardware components and the network connections for the FlexPod Datacenter for Accelerated RAG Pipeline with NVIDIA NIM design.

Figure 11. FlexPod Datacenter Physical Topology



Cisco UCS X-Series Direct
Cisco UCS S9108 Fabric Interconnect in Cisco UCS 9508 Chassis, Cisco UCS M7 Servers
Legend
100 Gbps Ethernet
vPC
vPC
Cisco Nexus 93600CD-GX
vPC
vPC
NetApp storage controllers AFF-C800
NetApp
NetApp

The software components of this solution consist of:

- Cisco Intersight SaaS platform to deploy, maintain, and support the FlexPod components
- Cisco Intersight Assist Virtual Appliance to connect NetApp ONTAP and Cisco Nexus switches to Cisco Intersight
- NetApp Active IQ Unified Manager to monitor and manage the storage and for NetApp ONTAP integration with Cisco Intersight
- Red Hat OCP to manage a Kubernetes containerized environment

FlexPod Datacenter with Red Hat OCP on Bare Metal

Red Hat OCP on Bare Metal Server Configuration

A simple Red Hat OCP cluster consists of at least five servers – 3 Master or Control Plane Nodes and 2 or more Worker Nodes where applications and VMs are run. In this lab validation 3 Worker Nodes were utilized. Based on OCP published requirements, the three Master Nodes were configured with 64GB RAM, and the three Worker Nodes were configured with 768GB RAM to handle containerized applications and VMs. Each Node was booted from M.2. Both a single M.2 module and 2 M.2 modules with RAID1 are supported. Also, the servers paired with X440p PCIe Nodes were configured as Workers. From a networking perspective, both the Masters and the Workers were configured with a single vNIC with Cisco UCS Fabric Failover in the Bare Metal or Management VLAN. The workers were configured with extra NICs (vNICs) to allow storage attachment to the Workers. Each worker had two additional vNICs with the iSCSI A and B VLANs configured as native to allow iSCSI persistent storage attachment and future iSCSI boot. These vNICs also had the NVMe-TCP A and B allowed VLANs assigned, allowing tagged VLAN interfaces for NVMe-TCP to be defined on the Workers. Finally, each worker had one additional vNIC with the OCP NFS VLAN configured as native to provide NFS persistent storage.

VLAN Configuration

Table 1 lists VLANs configured for setting up the FlexPod environment along with their usage.

Table 1. **VLAN Usage**

VLAN ID	Name	Usage	IP Subnets
2*	Native-VLAN	Use VLAN 2 as native VLAN instead of default VLAN (1)	
1020*	OOB-MGMT-VLAN	Out-of-band management VLAN to connect management ports for various devices	10.102.0.0/24 GW: 10.102.0.254
1022	OCP-BareMetal-MGMT	Routable OCP Bare Metal VLAN used for OCP cluster and node management	10.102.2.0/24 GW: 10.102.2.254
3012	OCP-iSCSI-A	Used for OCP iSCSI Persistent Storage	192.168.12.0/24
3022	OCP-iSCSI-B	Used for OCP iSCSI Persistent Storage	192.168.22.0/24
3032	OCP-NVMe-TCP-A	Used for OCP NVMe-TCP Persistent Storage	192.168.32.0/24
3042	OCP-NVMe-TCP-B	Used for OCP NVMe-TCP Persistent Storage	192.168.42.0/24
3052	OCP-NFS	Used for OCP NFS RWX Persistent Storage	192.168.52.0/24

Note: * VLAN configfigured in FlexPod Base.

Table 2 lists the VMs or bare metal servers necessary for deployment as outlined in this document.

Table 2. Virtual Machines

Virtual Machine Description	VLAN	IP Address	Comments
OCP AD1	1022	10.102.2.249	Hosted on pre-existing management infrastructure within the FlexPod
OCP AD2	1022	10.102.2.250	Hosted on pre-existing management infrastructure within the FlexPod
OCP Installer	1022	10.102.2.10	Hosted on pre-existing management infrastructure within the FlexPod
NetApp Active IQ Unified Manager	1021	10.102.1.97	Hosted on pre-existing management infrastructure within the FlexPod
Cisco Intersight Assist Virtual Appliance	1021	10.102.1.96	Hosted on pre-existing management infrastructure within the FlexPod

Software Revisions

Table 3 lists the software revisions for various components of the solution.

Table 3. Software Revisions

Layer	Device	Image Bundle	Comments
Compute	Cisco UCS Fabric Interconnect S9108	4.3(4.240078)	
	Cisco UCS X210C M7	5.2(2.240053)	
Network	Cisco Nexus 93600CD-GX NX-OS	10.3(4a)M	
Storage	NetApp AFF C800	ONTAP 9.14.1	Latest patch release
Software	Red Hat OCP	4.16	
	NetApp Astra Trident	24.06.1	
	NetApp DataOps Toolkit	2.5.0	
	Cisco Intersight Assist Appliance	1.0.9-675	1.0.9-538 initially installed and then automatically upgraded
	NetApp Active IQ Unified Manager	9.14	
	NVIDIA L40S GPU Driver	550.90.07	

FlexPod Cabling

The information in this section is provided as a reference for cabling the physical equipment in a FlexPod environment. To simplify cabling requirements, a cabling diagram was used.

The cabling diagram in this section contains the details for the prescribed and supported configuration of the NetApp AFF C800 running NetApp ONTAP 9.14.1.

Note: For any modifications of this prescribed architecture, consult the NetApp Interoperability Matrix Tool (IMT).

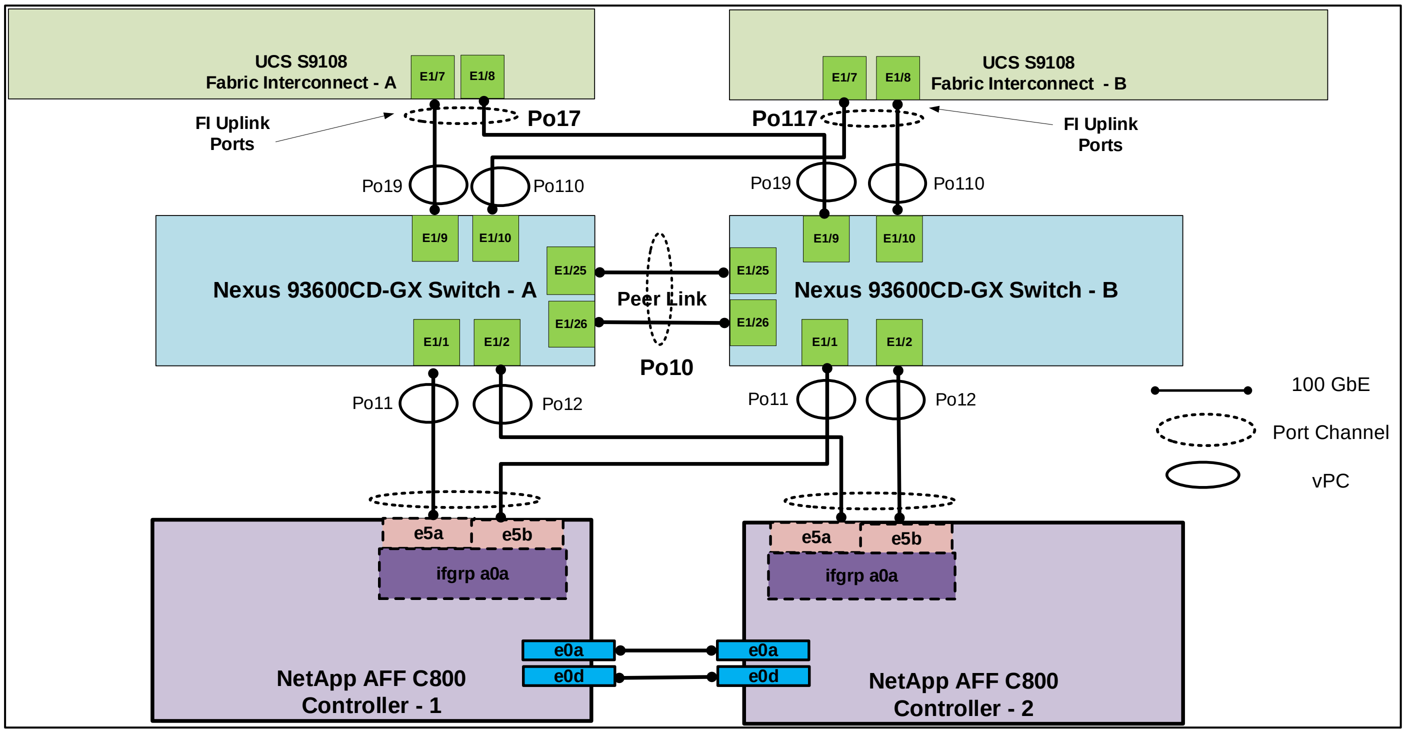
Note: This document assumes that out-of-band management ports are plugged into an existing management infrastructure at the deployment site. These interfaces will be used in various configuration steps.

Note: Be sure to use the cabling directions in this section as a guide.

The NetApp storage controller and disk shelves should be connected according to best practices for the specific storage controller and disk shelves. For disk shelf cabling, refer to NetApp Support.

Figure 12 details the cable connections used in the validation lab for the FlexPod topology based on the Cisco UCS S9108 Fabric Interconnect directly in the chassis. Two 100Gb links connect each Cisco UCS Fabric Interconnect to the Cisco Nexus Switches and each NetApp AFF controller to the Cisco Nexus Switches. Additional 1Gb management connections will be needed for one or more out-of-band network switches that sit apart from the FlexPod infrastructure. Each Cisco UCS Fabric Interconnect and Cisco Nexus switch is connected to the out-of-band network switches, and each AFF controller has a connection to the out-of-band network switches. Layer 3 network connectivity is required between the Out-of-Band (OOB) and In-Band (IB) Management Subnets.

Figure 12. FlexPod Cabling with Cisco UCS S9108 X-Series Direct Fabric Interconnects

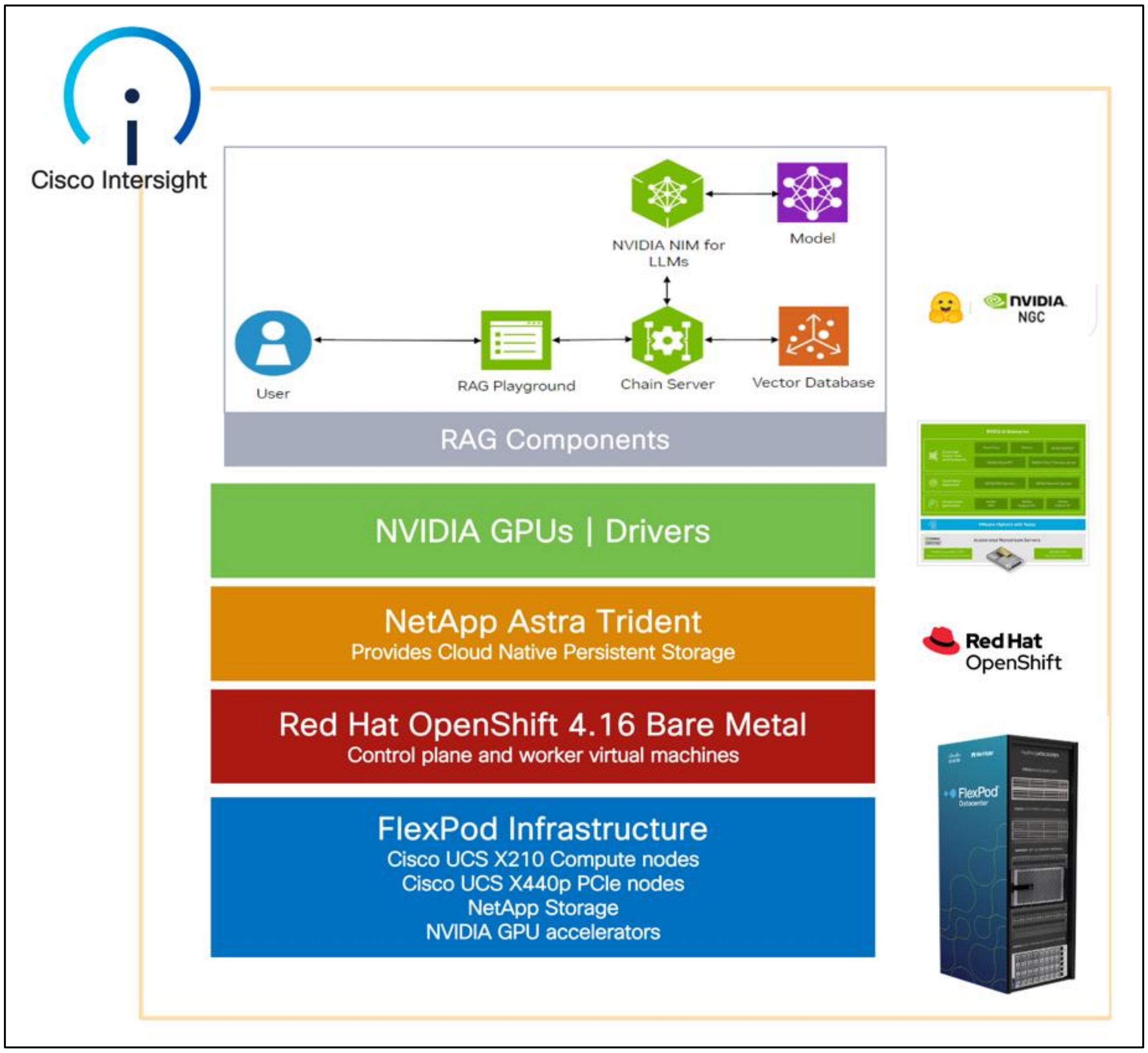


RAG Design Summary

Retrieval Augmented Generation (RAG) is designed to empower LLM solutions with real-time data access, preserving data privacy and mitigating LLM hallucinations. A typical RAG pipeline consists of several phases. The process of document ingestion occurs offline, and when an online query comes in, the retrieval of relevant documents and the generation of a response occurs. By using RAG, you can provide up-to-date and proprietary information with ease to LLMs and build a system that increases user trust, improves user experiences, and reduces hallucinations. NVIDIA RAG Enterprise designed with NVIDIA NeMo is an end-to-end platform for developing custom generative AI, anywhere. Deliver enterprise-ready models with precise data curation, cutting-edge customization, RAG, and accelerated performance.

The FlexPod architecture is designed as described in this CVD: FlexPod Datacenter with Red Hat OCP Bare Metal Manual Configuration with Cisco UCS X-Series Direct Deployment Guide. NVIDIA AI Enterprise and NVIDIA NIM microservices were installed to build a powerful platform for running Retrieval Augmented Generation. This layered approach, depicted below, was configured with best practices and security in mind, resulting in a high-performance, secure platform for Retrieval Augmented Generation. This platform can be extended for further AI applications such as Training, Fine Tuning, and other Inferencing use cases, provided that the platform is sized for the application.

Figure 13. RAG Components



Network Switch Configuration

This chapter contains the following:

- Physical Connectivity
- Cisco Nexus Switch Manual Configuration
- NetApp ONTAP Storage Configuration

This chapter provides a detailed procedure for configuring the Cisco Nexus 93600CD-GX switches for use in a FlexPod environment.

Note: The following procedures describe how to configure the Cisco Nexus switches for use in the OCP Bare Metal FlexPod environment. This procedure assumes the use of Cisco Nexus 9000 10.3(4a)M and includes the setup of NTP distribution on the bare metal VLAN. The interface-vlan feature and NTP commands are used in the setup. This procedure adds the tenant VLANs to the appropriate port-channels.

Physical Connectivity

Follow the physical connectivity guidelines for FlexPod as explained in section FlexPod Cabling.

Cisco Nexus Switch Manual Configuration

Procedure 1. Create Tenant VLANs on Cisco Nexus A and Cisco Nexus B

Step 1. Log into both Nexus switches as admin using ssh.

Step 2. Configure the OCP Bare Metal VLAN:

```
config t
vlan <bm-vlan-id for example, 1022>
name <tenant-name>-BareMetal-MGMT
```

Step 3. Configure OCP iSCSI VLANs:

```
vlan <iscsi-a-vlan-id for example, 3012>
name <tenant-name>-iSCSI-A
vlan <iscsi-b-vlan-id for example, 3022>
name <tenant-name>-iSCSI-B
```

Step 4. If configuring NVMe-TCP storage access, create the following two additional VLANs:

```
vlan <nvme-tcp-a-vlan-id for example, 3032>
name <tenant-name>-NVMe-TCP-A
vlan <nvme-tcp-b-vlan-id for example, 3042>
name <tenant-name>-NVMe-TCP-B
exit
```

Step 5. Add OCP NFS VLAN:

```
vlan <nfs-vlan-id for example, 3052>
name <tenant-name>-NFS
```

Step 6. Add VLANs to the vPC peer link in both Nexus switches:

```
int Po10
switchport trunk allowed vlan add
<bm-vlan-id>,<iscsi-a-vlan-id>,<iscsi-b-vlan-id>,<nvme-tcp-a-vlan-id>,<nvme-tcp-b-vlan-id>,<nfs-vlan-id>
```

Step 7. Add VLANs to the storage interfaces in both Nexus switches:

```
int Po11,Po12
switchport trunk allowed vlan add
<bm-vlan-id>,<iscsi-a-vlan-id>,<iscsi-b-vlan-id>,<nvme-tcp-a-vlan-id>,<nvme-tcp-b-vlan-id>,<nfs-vlan-id>
```

Step 8. Add VLANs to the UCS Fabric Interconnect Uplink interfaces in both Nexus switches:

```
int Po19,Po110
switchport trunk allowed vlan add
<bm-vlan-id>,<iscsi-a-vlan-id>,<iscsi-b-vlan-id>,<nvme-tcp-a-vlan-id>,<nvme-tcp-b-vlan-id>,<nfs-vlan-id>
```

Step 9. Add the Bare Metal VLAN to the Switch Uplink interface in both Nexus switches:

```
interface Po127
switchport trunk allowed vlan add <bm-vlan-id>
exit
```

Step 10. If configuring NTP Distribution in these Nexus Switches, add Tenant VRF and NTP Distribution Interface in Cisco Nexus A:

```
vrf context <tenant-name>
ip route 0.0.0.0/0 <bm-subnet-gateway>
exit
interface vlan<bm-vlan-id>
no shutdown
vrf member <tenant-name>
ip address <bm-switch-a-ntp-distr-ip>/<bm-vlan-mask-length>
exit
copy run start
```

Step 11. If configuring NTP Distribution in these Nexus Switches, add Tenant VRF and NTP Distribution Interface in Cisco Nexus B:

```
vrf context <tenant-name>
ip route 0.0.0.0/0 <bm-subnet-gateway>
exit
interface vlan<bm-vlan-id>
no shutdown
vrf member <tenant-name>
ip address <bm-switch-b-ntp-distr-ip>/<bm-vlan-mask-length>
exit
copy run start
```

Step 12. The following commands can be used to see the switch configuration and status:

```
show run
show vpc
show vlan
show port-channel summary
show ntp peer-status
show cdp neighbors
show lldp neighbors
show run int
show int
show udld neighbors
show int status
```

NetApp ONTAP Storage Configuration

Procedure 1. Configure the NetApp ONTAP storage

Step 1. Log into the Cluster.

Step 2. Open an SSH connection to either the cluster IP or the host name.

Step 3. Log into the admin user with the password you provided earlier.

Procedure 2. Configure the NetApp ONTAP Storage for the OCP Tenant

Note: By default, all network ports are included in a separate default broadcast domain. Network ports used for data services (for example, e5a, e5b, and so on) should be removed from their default broadcast domain and that broadcast domain should be deleted.

Step 1. Delete any Default-N automatically created broadcast domains:

```
network port broadcast-domain delete -broadcast-domain <Default-N> -ipspace Default
network port broadcast-domain show
```

Note: Delete the Default broadcast domains with Network ports (Default-1, Default-2, and so on). This does not include Cluster ports and management ports.

Step 2. Create an IPspace for the OCP tenant:

```
network ipspace create -ipspace AA02-OCP
```

Step 3. Create the OCP-MGMT, OCP-iSCSI-A, OCP-iSCSI-B, OCP-NVMe-TCP-A , OCP-NVMe-TCP-B, and OCP-NFS broadcast domains with appropriate maximum transmission unit (MTU):

```
network port broadcast-domain create -broadcast-domain OCP-MGMT -mtu 1500 -ipspace AA02-OCP
network port broadcast-domain create -broadcast-domain OCP-iSCSI-A -mtu 9000 -ipspace AA02-OCP
network port broadcast-domain create -broadcast-domain OCP-iSCSI-B -mtu 9000 -ipspace AA02-OCP
network port broadcast-domain create -broadcast-domain OCP-NVMe-TCP-A -mtu 9000 -ipspace AA02-OCP
network port broadcast-domain create -broadcast-domain OCP-NVMe-TCP-B -mtu 9000 -ipspace AA02-OCP
network port broadcast-domain create -broadcast-domain OCP-NFS -mtu 9000 -ipspace AA02-OCP
```

Step 4. Create the OCP management VLAN ports and add them to the OCP management broadcast domain:

```
network port vlan create -node AA02-C800-01 -vlan-name a0a-1022
network port vlan create -node AA02-C800-02 -vlan-name a0a-1022
network port broadcast-domain add-ports -ipspace AA02-OCP -broadcast-domain OCP-MGMT -ports
AA02-C800-01:a0a-1022,AA02-C800-02:a0a-1022
```

Step 5. Create the OCP iSCSI VLAN ports and add them to the OCP iSCSI broadcast domains:

```
network port vlan create -node AA02-C800-01 -vlan-name a0a-3012
network port vlan create -node AA02-C800-02 -vlan-name a0a-3012
network port broadcast-domain add-ports -ipspace AA02-OCP -broadcast-domain OCP-iSCSI-A -ports
AA02-C800-01:a0a-3012,AA02-C800-02:a0a-3012

network port vlan create -node AA02-C800-01 -vlan-name a0a-3022
network port vlan create -node AA02-C800-02 -vlan-name a0a-3022
network port broadcast-domain add-ports -ipspace AA02-OCP -broadcast-domain OCP-iSCSI-B -ports
AA02-C800-01:a0a-3022,AA02-C800-02:a0a-3022
```

Step 6. Create the OCP NVMe-TCP VLAN ports and add them to the OCP NVMe-TCP broadcast domains:

```
network port vlan create -node AA02-C800-01 -vlan-name a0a-3032
network port vlan create -node AA02-C800-02 -vlan-name a0a-3032
network port broadcast-domain add-ports -ipspace AA02-OCP -broadcast-domain OCP-NVMe-TCP-A -ports
AA02-C800-01:a0a-3032,AA02-C800-02:a0a-3032

network port vlan create -node AA02-C800-01 -vlan-name a0a-3042
network port vlan create -node AA02-C800-02 -vlan-name a0a-3042
network port broadcast-domain add-ports -ipspace AA02-OCP -broadcast-domain OCP-NVMe-TCP-B -ports
AA02-C800-01:a0a-3042,AA02-C800-02:a0a-3042
```

Step 7. Create the OCP NFS VLAN ports and add them to the OCP NFS broadcast domain:

```
network port vlan create -node AA02-C800-01 -vlan-name a0a-3052
network port vlan create -node AA02-C800-02 -vlan-name a0a-3052
network port broadcast-domain add-ports -ipspace AA02-OCP -broadcast-domain OCP-NFS -ports
AA02-C800-01:a0a-3052,AA02-C800-02:a0a-3052
```

Step 8. Create the SVM (Storage Virtual Machine) in the IPspace. Run the `vserver create` command:

```
vserver create -vserver OCP-Trident-SVM -ipspace AA02-OCP
```

Note: The SVM must be created in the IPspace. An SVM cannot be moved into an IPspace later.

Step 9. Add the required data protocols to the SVM and remove the unused data protocols from the SVM:

```
vserver add-protocols -vserver OCP-Trident-SVM -protocols iscsi,nfs,nvme
vserver remove-protocols -vserver OCP-Trident-SVM -protocols cifs,fcp,s3
```

Step 10. Add the two data aggregates to the OCP-Trident-SVM aggregate list and enable and run the NFS protocol in the SVM:

```
vserver modify -vserver OCP-Trident-SVM -aggr-list AA02_C800_01_SSD_CAP_1,AA02_C800_02_SSD_CAP_1
vserver nfs create -vserver OCP-Trident-SVM -udp disabled -v3 enabled -v4.1 enabled
```

Step 11. Create a Load-Sharing Mirror of the SVM Root Volume. Create a volume to be the load-sharing mirror of the infrastructure SVM root volume only on the node that does not have the Root Volume:

```
volume show -vserver OCP-Trident-SVM # Identify the aggregate and node where the vserver root volume is located.
volume create -vserver OCP-Trident-SVM -volume OCP_Trident_SVM_root_lsm01 -aggregate AA02_C800_0<x>_SSD_CAP_1
-size 1GB -type DP # Create the mirror volume on the other node
```

Step 12. Create the 15min interval job schedule:

```
job schedule interval create -name 15min -minutes 15
```

Step 13. Create the mirroring relationship:

```
snapmirror create -source-path OCP-Trident-SVM:OCP_Trident_SVM_root -destination-path
OCP-Trident-SVM:OCP_Trident_SVM_root_lsm01 -type LS -schedule 15min
```

Step 14. Initialize and verify the mirroring relationship:

```
snapmirror initialize-ls-set -source-path OCP-Trident-SVM:OCP_Trident_SVM_root

snapmirror show -vserver OCP-Trident-SVM

                                                                       Progress
Source            Destination Mirror  Relationship   Total             Last
Path        Type  Path        State   Status         Progress  Healthy Updated
----------- ---- ------------ ------- -------------- --------- ------- --------
AA02-C800://OCP-Trident-SVM/OCP_Trident_SVM_root
            LS   AA02-C800://OCP-Trident-SVM/OCP_Trident_SVM_root_lsm01
                              Snapmirrored
                                      Idle           -         true    -
```

Step 15. Create the iSCSI and NVMe services:

```
vserver iscsi create -vserver OCP-Trident-SVM -status-admin up
vserver iscsi show -vserver OCP-Trident-SVM

                 Vserver: OCP-Trident-SVM
             Target Name: iqn.1992-08.com.netapp:sn.8442b0854ebb11efb1a7d039eab7b2f3:vs.5
            Target Alias: OCP-Trident-SVM
   Administrative Status: up

vserver nvme create -vserver OCP-Trident-SVM -status-admin up
vserver nvme show -vserver OCP-Trident-SVM

           Vserver Name: OCP-Trident-SVM
  Administrative Status: up
Discovery Subsystem NQN: nqn.1992-08.com.netapp:sn.8442b0854ebb11efb1a7d039eab7b2f3:discovery
```

Note: Make sure licenses are installed for all storage protocols used before creating the services.

Step 16. To create the login banner for the SVM, run the following command:

```
security login banner modify -vserver OCP-Trident-SVM -message "This OCP-Trident-SVM is reserved for authorized
users only!"
```

Step 17. Remove insecure ciphers from the SVM. Ciphers with the suffix CBC are considered insecure. To remove the CBC ciphers, run the following NetApp ONTAP command:

```
security ssh remove -vserver OCP-Trident-SVM -ciphers aes256-cbc,aes192-cbc,aes128-cbc,3des-cbc
```

Step 18. Create a new rule for the SVM NFS subnet in the default export policy and assign the policy to the SVM's root volume:

```
vserver export-policy rule create -vserver OCP-Trident-SVM -policyname default -ruleindex 1 -protocol nfs
-clientmatch 192.168.52.0/24 -rorule sys -rwrule sys -superuser sys -allow-suid true

volume modify –vserver OCP-Trident-SVM –volume OCP_Trident_SVM_root –policy default
```

Step 19. Create and enable the audit log in the SVM:

```
volume create -vserver OCP-Trident-SVM -volume audit_log -aggregate AA02_C800_01_SSD_CAP_1 -size 50GB -state
online -policy default -junction-path /audit_log -space-guarantee none -percent-snapshot-space 0

snapmirror update-ls-set -source-path OCP-Trident-SVM:OCP_Trident_SVM_root
vserver audit create -vserver OCP-Trident-SVM -destination /audit_log
vserver audit enable -vserver OCP-Trident-SVM
```

Step 20. Run the following commands to create NFS Logical Interfaces (LIFs):

```
network interface create -vserver OCP-Trident-SVM -lif nfs-lif-01 -service-policy default-data-files -home-node
AA02-C800-01 -home-port a0a-3052 -address 192.168.52.51 -netmask 255.255.255.0 -status-admin up -failover-policy
broadcast-domain-wide -auto-revert true

network interface create -vserver OCP-Trident-SVM -lif nfs-lif-02 -service-policy default-data-files -home-node
AA02-C800-02 -home-port a0a-3052 -address 192.168.52.52 -netmask 255.255.255.0 -status-admin up -failover-policy
broadcast-domain-wide -auto-revert true
```

Step 21. Run the following commands to create iSCSI LIFs:

```
network interface create -vserver OCP-Trident-SVM -lif iscsi-lif-01a -service-policy default-data-iscsi
-home-node AA02-C800-01 -home-port a0a-3012 -address 192.168.12.51 -netmask 255.255.255.0 -status-admin up

network interface create -vserver OCP-Trident-SVM -lif iscsi-lif-01b -service-policy default-data-iscsi
-home-node AA02-C800-01 -home-port a0a-3022 -address 192.168.22.51 -netmask 255.255.255.0 -status-admin up

network interface create -vserver OCP-Trident-SVM -lif iscsi-lif-02a -service-policy default-data-iscsi
-home-node AA02-C800-02 -home-port a0a-3012 -address 192.168.12.52 -netmask 255.255.255.0 -status-admin up

network interface create -vserver OCP-Trident-SVM -lif iscsi-lif-02b -service-policy default-data-iscsi
-home-node AA02-C800-02 -home-port a0a-3022 -address 192.168.22.52 -netmask 255.255.255.0 -status-admin up
```

Step 22. Run the following commands to create NVMe-TCP LIFs:

```
network interface create -vserver OCP-Trident-SVM -lif nvme-tcp-lif-01a -service-policy default-data-nvme-tcp
-home-node AA02-C800-01 -home-port a0a-3032 -address 192.168.32.51 -netmask 255.255.255.0 -status-admin up

network interface create -vserver OCP-Trident-SVM -lif nvme-tcp-lif-01b -service-policy default-data-nvme-tcp
-home-node AA02-C800-01 -home-port a0a-3042 -address 192.168.42.51 -netmask 255.255.255.0 -status-admin up

network interface create -vserver OCP-Trident-SVM -lif nvme-tcp-lif-02a -service-policy default-data-nvme-tcp
-home-node AA02-C800-02 -home-port a0a-3032 -address 192.168.32.52 -netmask 255.255.255.0 -status-admin up

network interface create -vserver OCP-Trident-SVM -lif nvme-tcp-lif-02b -service-policy default-data-nvme-tcp
-home-node AA02-C800-02 -home-port a0a-3042 -address 192.168.42.52 -netmask 255.255.255.0 -status-admin up
```

Step 23. Run the following command to create the SVM-MGMT LIF:

```
network interface create -vserver OCP-Trident-SVM -lif svm-mgmt -service-policy default-management -home-node
AA02-C800-01 -home-port a0a-1022 -address 10.102.2.50 -netmask 255.255.255.0 -status-admin up -failover-policy
broadcast-domain-wide -auto-revert true
```

Step 24. Run the following command to verify LIFs:

```
network interface show -vserver OCP-Trident-SVM
            Logical    Status     Network            Current       Current Is
Vserver     Interface  Admin/Oper Address/Mask       Node          Port    Home
----------- ---------- ---------- ------------------ ------------- ------- ----
OCP-Trident-SVM
            iscsi-lif-01a
                         up/up    192.168.12.51/24   AA02-C800-01  a0a-3012
                                                                           true
            iscsi-lif-01b
                         up/up    192.168.22.51/24   AA02-C800-01  a0a-3022
                                                                           true
            iscsi-lif-02a
                         up/up    192.168.12.52/24   AA02-C800-02  a0a-3012
                                                                           true
            iscsi-lif-02b
                         up/up    192.168.22.52/24   AA02-C800-02  a0a-3022
                                                                           true
            nfs-lif-01   up/up    192.168.52.51/24   AA02-C800-01  a0a-3052
                                                                           true
            nfs-lif-02   up/up    192.168.52.52/24   AA02-C800-02  a0a-3052
                                                                           true
            nvme-tcp-lif-01a
                         up/up    192.168.32.51/24   AA02-C800-01  a0a-3032
                                                                           true
            nvme-tcp-lif-01b
                         up/up    192.168.42.51/24   AA02-C800-01  a0a-3042
                                                                           true
            nvme-tcp-lif-02a
                         up/up    192.168.32.52/24   AA02-C800-02  a0a-3032
                                                                           true
            nvme-tcp-lif-02b
                         up/up    192.168.42.52/24   AA02-C800-02  a0a-3042
                                                                           true
            svm-mgmt     up/up    10.102.2.50/24     AA02-C800-01  a0a-1022
                                                                           true
11 entries were displayed.
```

Step 25. Create a default route that enables the SVM management interface to reach the outside world:

```
network route create -vserver OCP-Trident-SVM -destination 0.0.0.0/0 -gateway 10.102.2.254
```

Step 26. Set a password for the SVM vsadmin user and unlock the user:

```
security login password -username vsadmin -vserver OCP-Trident-SVM
Enter a new password:
Enter it again:

security login unlock -username vsadmin -vserver OCP-Trident-SVM
```

Step 27. Add the OCP DNS servers to the SVM:

```
dns create -vserver OCP-Trident-SVM -domains ocp.flexpodb4.cisco.com -name-servers 10.102.2.249,10.102.2.250
```

Cisco Intersight Managed Mode Configuration

This chapter contains the following:

The Cisco Intersight platform is a management solution delivered as a service with embedded analytics for Cisco and third-party IT infrastructures. The Cisco Intersight Managed Mode (also referred to as Cisco IMM or Intersight Managed Mode) is an architecture that manages Cisco Unified Computing System (Cisco UCS) Fabric Interconnect-attached systems through a Redfish-based standard model. Cisco Intersight managed mode standardizes both policy and operation management for Cisco UCS X210c M7 compute nodes used in this deployment guide.

Cisco UCS B-Series M5, M6, M7 blades and C-Series M6 and M7 servers, connected and managed through Cisco UCS 6400 & 6500 Fabric Interconnects, are also supported by IMM. For a complete list of supported platforms, go to:
https://www.cisco.com/c/en/us/td/docs/unified_computing/Intersight/b_Intersight_Managed_Mode_Configuration_Guide/b_intersight_managed_mode_guide_chapter_01010.html

Procedure 1. Set up Cisco Intersight Resource Group

In this procedure, a Cisco Intersight resource group for the Red Hat OCP tenant is created where resources such as targets will be logically grouped. In this deployment, a single resource group is created to host all the resources, but you can choose to create multiple resource groups for granular control of the resources.

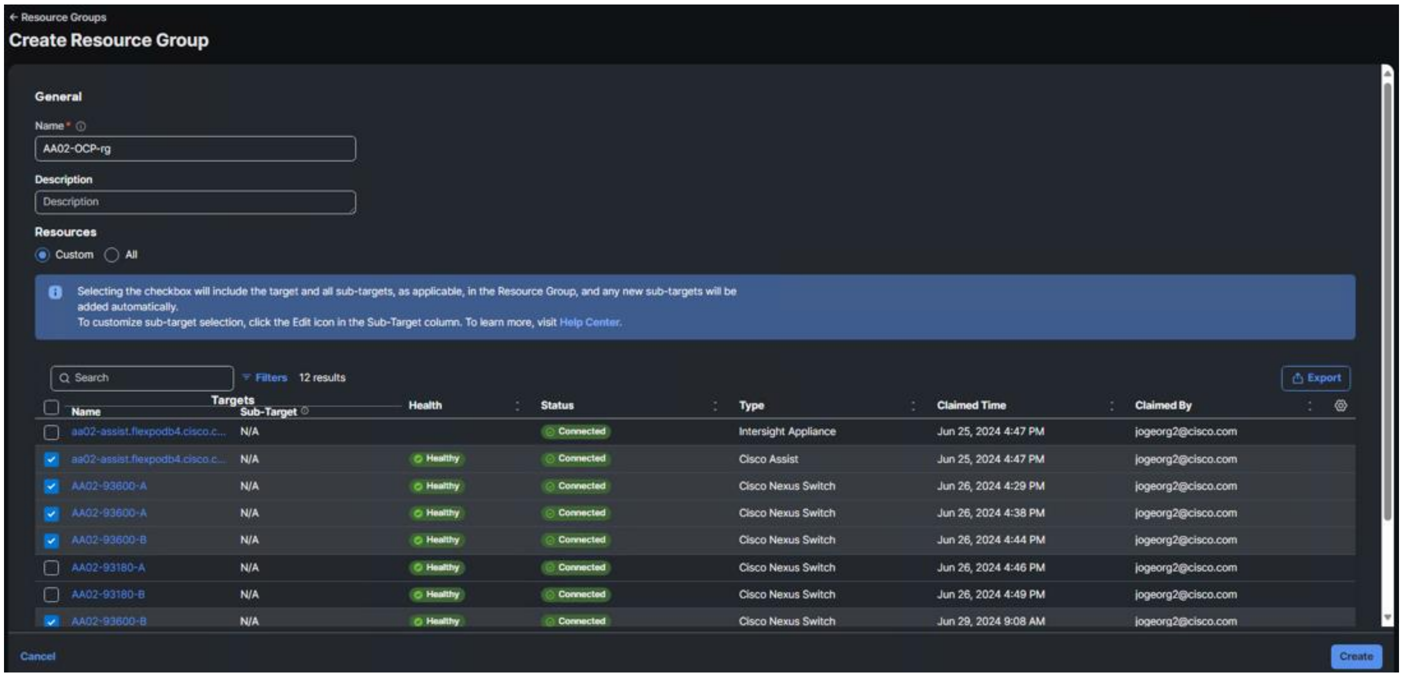
Step 1. Log into **Cisco Intersight**.

Step 2. Select **System**. On the left, click **Settings** (the gear icon).

Step 3. Click **Resource Groups** in the middle panel.

Step 4. Click **+ Create Resource Group** in the top-right corner.

Step 5. Provide a name for the Resource Group (for example, AA02-OCP-rg).



Name * : AA02-OCP-rg

Description : Description

Resources: [✓ Custom] All

Selecting the checkbox will include the target and all sub-targets, as applicable, in the Resource Group, and any new sub-targets will be added automatically.
To customize sub-target selection, click the Edit icon in the Sub-Target column. To learn more, visit Help Center.

12 results

	Targets Name	Sub-Target	Health	Status	Type	Claimed Time	Claimed By
☐	aa02-assist.flexpodb4.cisco.c...	N/A		Connected	Intersight Appliance	Jun 25, 2024 4:47 PM	jogeorg2@cisco.com
✓	aa02-assist.flexpodb4.cisco.c...	N/A	Healthy	Connected	Cisco Assist	Jun 25, 2024 4:47 PM	jogeorg2@cisco.com
✓	AA02-93600-A	N/A	Healthy	Connected	Cisco Nexus Switch	Jun 26, 2024 4:29 PM	jogeorg2@cisco.com
✓	AA02-93600-A	N/A	Healthy	Connected	Cisco Nexus Switch	Jun 26, 2024 4:38 PM	jogeorg2@cisco.com
✓	AA02-93600-B	N/A	Healthy	Connected	Cisco Nexus Switch	Jun 26, 2024 4:44 PM	jogeorg2@cisco.com
☐	AA02-93180-A	N/A	Healthy	Connected	Cisco Nexus Switch	Jun 26, 2024 4:46 PM	jogeorg2@cisco.com
☐	AA02-93180-B	N/A	Healthy	Connected	Cisco Nexus Switch	Jun 26, 2024 4:49 PM	jogeorg2@cisco.com
✓	AA02-93600-B	N/A	Healthy	Connected	Cisco Nexus Switch	Jun 29, 2024 9:08 AM	jogeorg2@cisco.com

Step 6. Under Resources, select **Custom**.

Step 7. Select all resources that are connected to this Red Hat OCP FlexPod.

Step 8. Click **Create.**

Procedure 2. Set Up Cisco Intersight Organization

In this procedure, an Intersight organization for the Red Hat OCP tenant is created where all Cisco Intersight Managed Mode configurations including policies are defined.

Step 1. Log into the **Cisco Intersight** portal.

Step 2. Select **System.** On the left, click **Settings** (the gear icon).

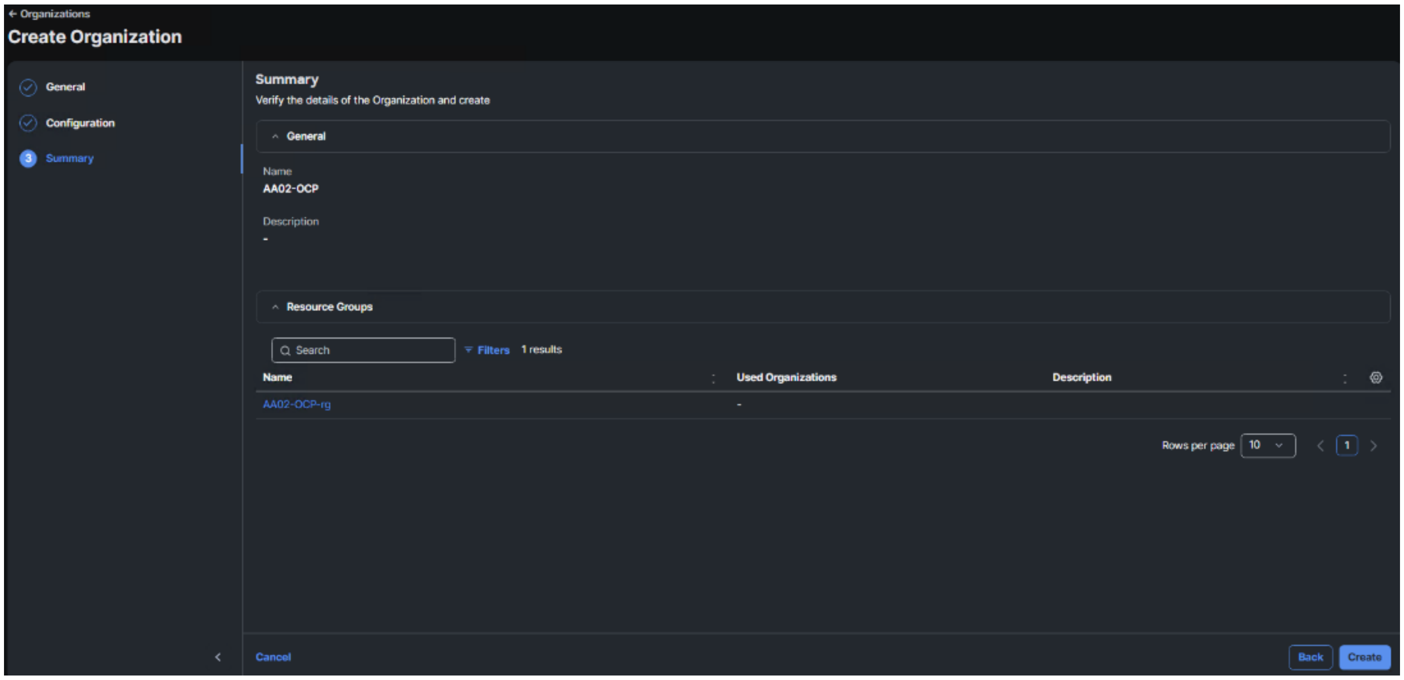
Step 3. Click **Organizations** in the middle panel.

Step 4. Click **+ Create Organization** in the top-right corner.

Step 5. Provide a name for the organization (for example, AA02-OCP), optionally select Share Resources with Other Organizations, and click **Next.**

Step 6. Select the Resource Group created in the last step (for example, AA02-OCP-rg) and click **Next.**

Step 7. Click **Create.**



Summary
Verify the details of the Organization and create

General

Name
AA02-OCP

Description
-

Resource Groups

1 results

Name	Used Organizations	Description
AA02-OCP-rg	-	

Procedure 3. Add OCP VLANs to VLAN Policy

Step 1. Log into the **Cisco Intersight** portal.

Step 2. Select **Infrastructure Service**. On the left, select **Profiles** then under Profiles select **UCS Domain Profiles**.

Step 3. To the right of the UCS Domain Profile used for the OCP tenant, click **...** and select **Edit.**

Step 4. Click **Next** to go to UCS Domain Assignment.

Step 5. Click **Next** to go to VLAN & VSAN Configuration.

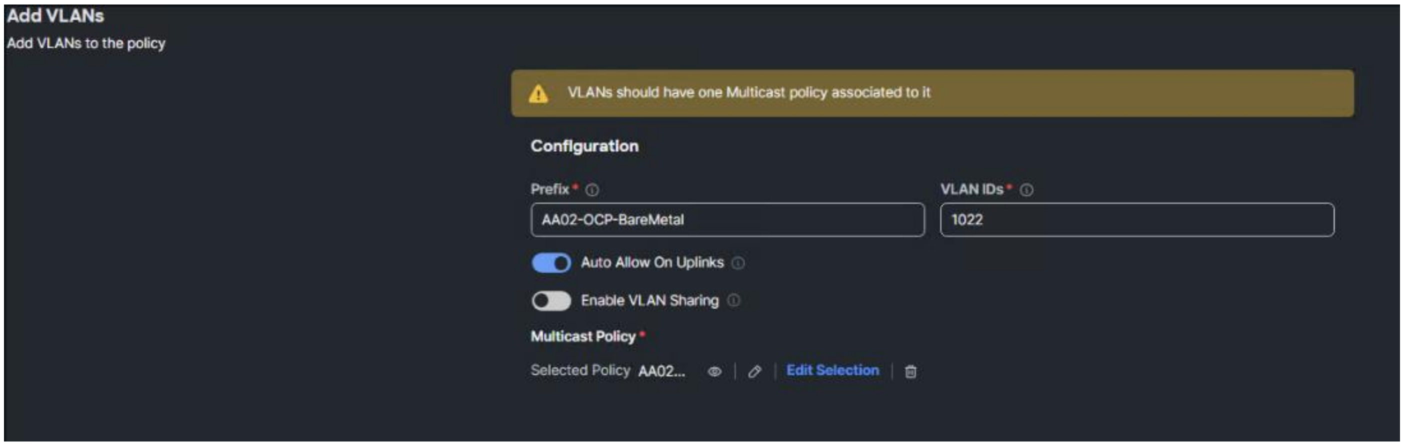
Step 6. Under VLAN & VSAN Configuration, click the pencil icon to the left of the VLAN Policy to Edit the policy.

Step 7. Click **Next** to go to Policy Details.

Step 8. To add the OCP-BareMetal VLAN, click **Add VLANs**.

Step 9. For the Prefix, enter the VLAN name. For the VLAN ID, enter the VLAN id. Leave Auto Allow on Uplinks enabled and Enable VLAN Sharing disabled.

Step 10. Under Multicast Policy, click **Select Policy** and select the already configured Multicast Policy (for example, AA02-MCAST).



Add VLANs

Add VLANs to the policy

VLANs should have one Multicast policy associated to it

Configuration

Prefix *: AA02-OCP-BareMetal

VLAN IDs *: 1022

Auto Allow On Uplinks: enabled

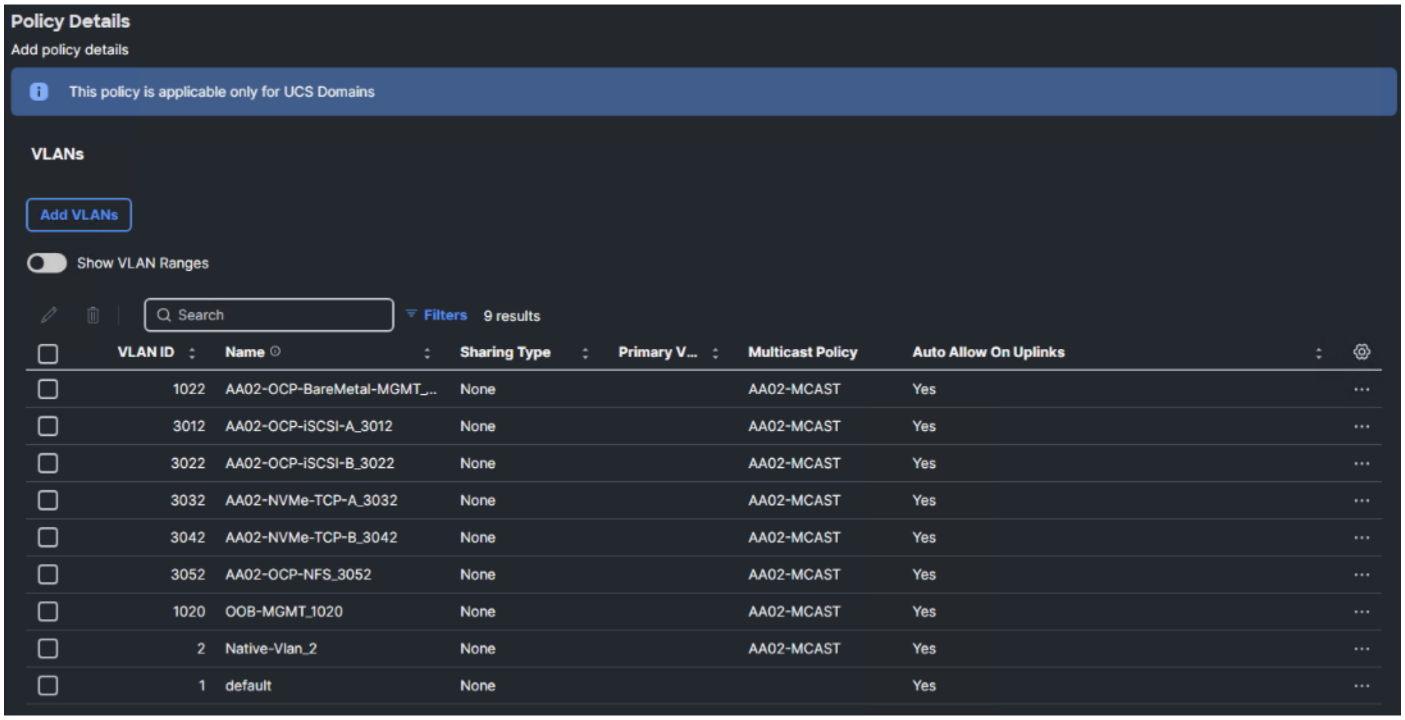
Enable VLAN Sharing: disabled

Multicast Policy *

Selected Policy AA02...

Step 11. Click **Add** to add the VLAN to the policy.

Step 12. Repeat step 10 and step 11 to add all the VLANs in Table 1 to the VLAN Policy.



Policy Details

Add policy details

This policy is applicable only for UCS Domains

VLANs

9 results

VLAN ID	Name	Sharing Type	Primary V...	Multicast Policy	Auto Allow On Uplinks
1022	AA02-OCP-BareMetal-MGMT_...	None		AA02-MCAST	Yes
3012	AA02-OCP-iSCSI-A_3012	None		AA02-MCAST	Yes
3022	AA02-OCP-iSCSI-B_3022	None		AA02-MCAST	Yes
3032	AA02-NVMe-TCP-A_3032	None		AA02-MCAST	Yes
3042	AA02-NVMe-TCP-B_3042	None		AA02-MCAST	Yes
3052	AA02-OCP-NFS_3052	None		AA02-MCAST	Yes
1020	OOB-MGMT_1020	None		AA02-MCAST	Yes
2	Native-Vlan_2	None		AA02-MCAST	Yes
1	default	None			Yes

Step 13. Click **Save** to save the VLAN Policy.

Step 14. Click **Next** three times to get to the UCS Domain Profile Summary page.

Step 15. Click **Deploy** and then **Deploy** again to deploy the UCS Domain Profile.

Cisco UCS IMM Manual Configuration

Configure Server Profile Template

In the Cisco Intersight platform, a server profile enables resource management by simplifying policy alignment and server configuration. The server profiles are derived from a server profile template. A Server profile template and its associated policies can be created using the server profile template wizard. After creating the server profile template, customers can derive multiple consistent server profiles from the template.

The server profile templates captured in this deployment guide supports Cisco UCS X210c M7 compute nodes with 5th Generation VICs and can be modified to support other Cisco UCS blades and rack mount servers.

vNIC and vHBA Placement for Server Profile Template

In this deployment, separate server profile templates are created for OCP Worker and Master Nodes where Worker Nodes have storage network interfaces to support workloads, but Master Nodes do not. The vNIC layout is explained below. While most of the policies are common across various templates, the LAN connectivity policies are unique and will use the information in the following tables:

- 1 vNIC is configured for OCP Master Nodes. This vNIC is manually placed as listed in Table 4.
- 4 vNICs are configured for OCP Worker Nodes. These vNICs are manually placed as listed in Table 5. NVMe-TCP VLAN Interfaces can be added as tagged VLANs to the iSCSI vNICs when NVMe-TCP is being used.

Table 4. vNIC placement for OCP Master Nodes

vNIC/vHBA Name	Switch ID	PCI Order	Fabric Failover	Native VLAN	Allowed VLANs
eno5	A	0	Y	OCP-BareMetal	OCP-BareMetal-MGMT

Table 5. vNIC placement for OCP Worker Nodes

vNIC/vHBA Name	Switch ID	PCI Order	Fabric Failover	Native VLAN	Allowed VLANs
eno5	A	0	Y	OCP-BareMetal	OCP-BareMetal-MGMT
eno6	A	1	N	OCP-iSCSI-A	OCP-iSCSI-A, OCP-NVMe-TCP-A
eno7	B	2	N	OCP-iSCSI-B	OCP-iSCSI-B, OCP-NVMe-TCP-B

vNIC/vHBA Name	Switch ID	PCI Order	Fabric Failover	Native VLAN	Allowed VLANs
eno8	A	3	N	OCP-NFS	OCP-NFS

Note: OCP-NVMe-TCP-A will be added to eno6 as a VLAN interface. OCP-NVMe-TCP-B will be added to eno7 as a VLAN interface.

Procedure 1. Create Master Node Server Profile Template

A Server Profile Template will first be created for the OCP Master Nodes. This procedure will assume an X210C M7 is being used but can be modified for other server types.

Step 1. Log into **Cisco Intersight**.

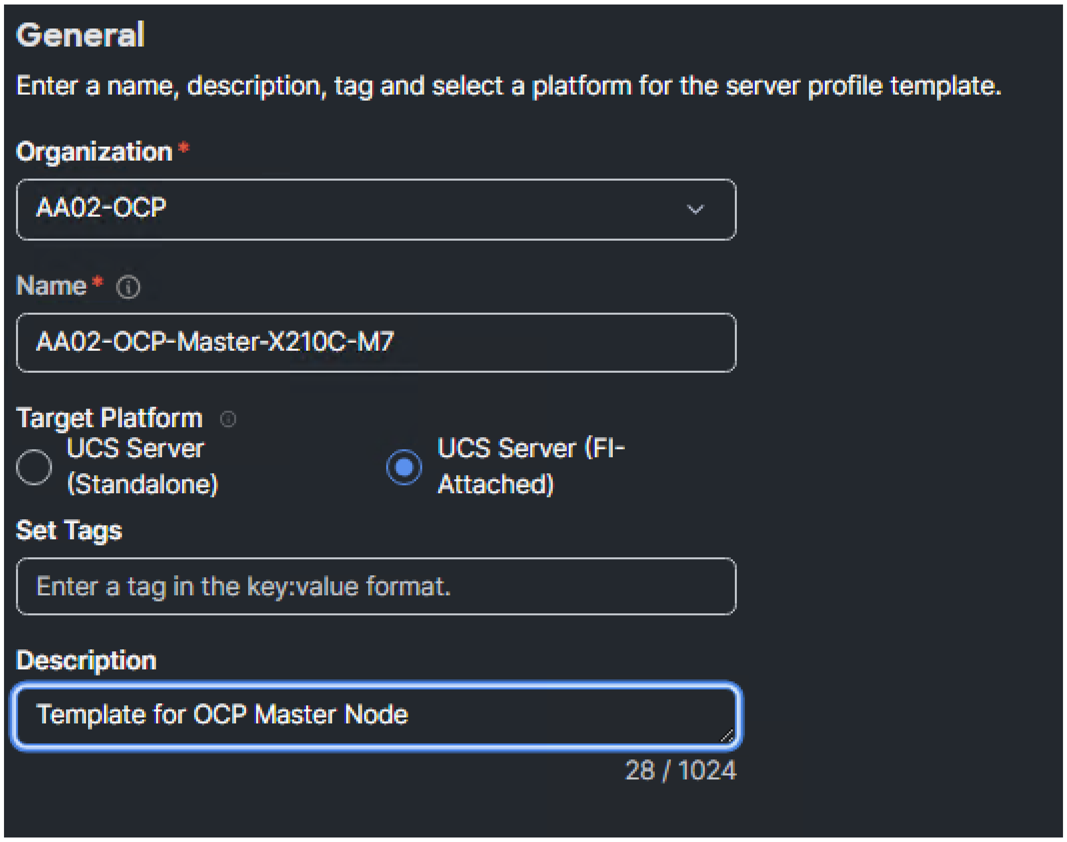
Step 2. Go to **Infrastructure Service** > **Configure** > **Templates** and in the main window and select UCS Server Profile Templates. Click **Create UCS Server Profile Template**.

Step 3. Select the organization from the drop-down list (for example, AA02-OCP).

Step 4. Provide a name for the server profile template (for example, AA02-OCP-Master-X210C-M7)

Step 5. Select **UCS Server (FI-Attached)**.

Step 6. Provide an optional description.


General

Enter a name, description, tag and select a platform for the server profile template.

Organization *: AA02-OCP

Name *: AA02-OCP-Master-X210C-M7

Target Platform: UCS Server (Standalone) [✓ UCS Server (FI-Attached)]

Set Tags: Enter a tag in the key:value format.

Description: Template for OCP Master Node

28 / 1024

Step 7. Click **Next**.

Compute Configuration

Procedure 2. Compute Configuration – Configure UUID Pool

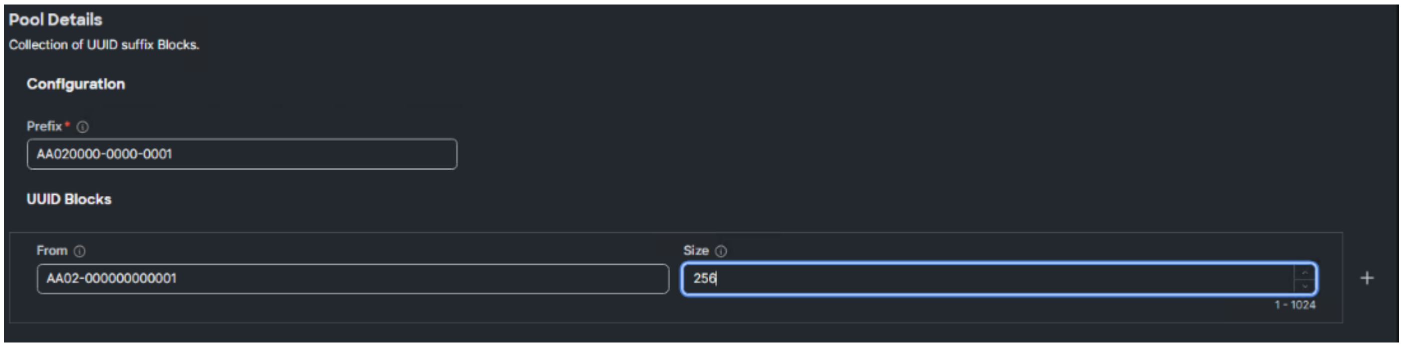
Step 1. Click **Select Pool** under UUID Pool and then in the pane on the right, click **Create New**.

Step 2. Verify correct organization is selected from the drop-down list (for example, AA02) and provide a name for the UUID Pool (for example, AA02-OCP-UUID-Pool).

Step 3. Provide an optional Description and click **Next.**

Step 4. Provide a unique UUID Prefix (for example, a prefix of AA020000-0000-0001 was used).

Step 5. Add a UUID block.



Pool Details
Collection of UUID suffix Blocks.

Configuration

Prefix * : AA020000-0000-0001

UUID Blocks

From: AA02-000000000001
Size: 256 (1 - 1024)

Step 6. Click **Create.**

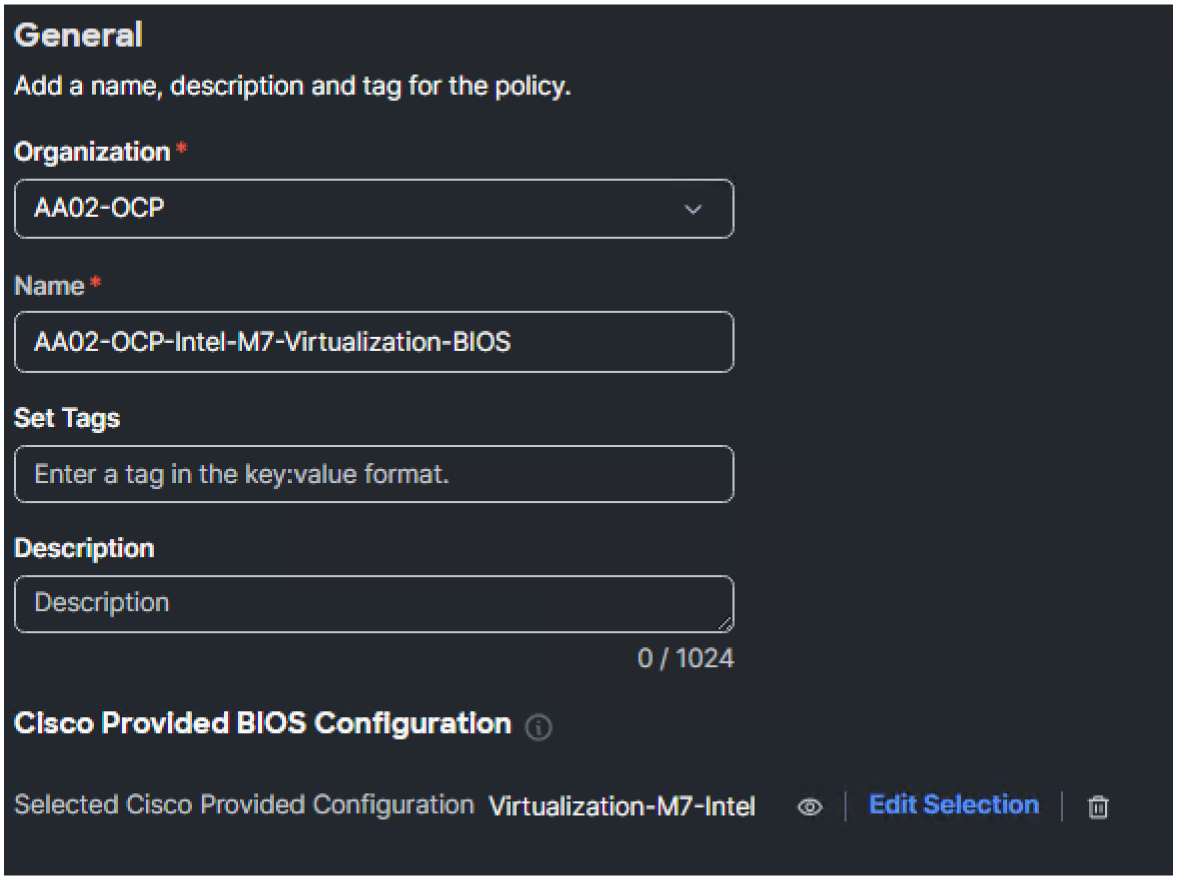
Procedure 3. Configure BIOS Policy

Step 1. Click **Select Policy** next to BIOS and in the pane on the right, click **Create New**.

Step 2. Verify correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-Intel-M7-Virtualization-BIOS).

Step 3. Enter an optional Description.

Step 4. Click **Select Cisco Provided Configuration**. In the Search box, type **Vir**. Select **Virtualization-M7-Intel** or the appropriate Cisco Provided Configuration for your platform.


General

Add a name, description and tag for the policy.

Organization *

AA02-OCP

Name *

AA02-OCP-Intel-M7-Virtualization-BIOS

Set Tags

Enter a tag in the key:value format.

Description

Description

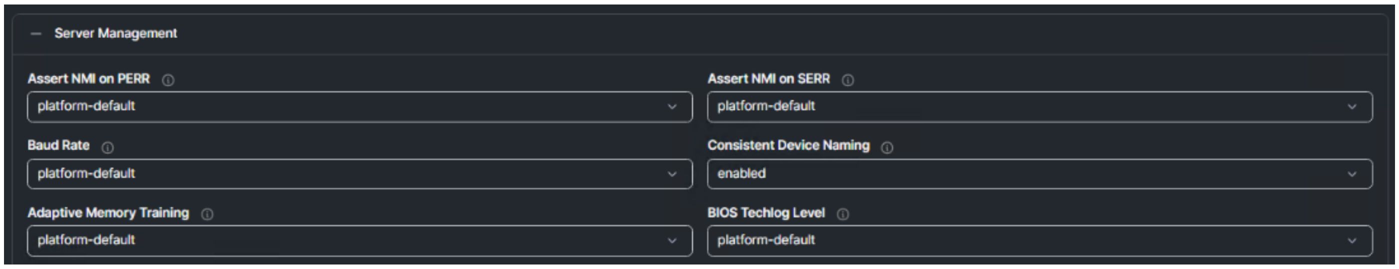
0 / 1024

Cisco Provided BIOS Configuration

Selected Cisco Provided Configuration Virtualization-M7-Intel Edit Selection

Step 5. Click **Next.**

Step 6. On the Policy Details screen, expand **Server Management**. From the drop-down list select **enabled** for the **Consistent Device Naming** BIOS token.


Server Management

Assert NMI on PERR: platform-default

Assert NMI on SERR: platform-default

Baud Rate: platform-default

Consistent Device Naming: enabled

Adaptive Memory Training: platform-default

BIOS Techlog Level: platform-default

Note: The BIOS Policy settings specified here are from the [Performance Tuning Best Practices Guide for Cisco UCS M7 Platforms - Cisco] with the Virtualization workload. For other platforms, the appropriate documents are listed below:

- [Performance Tuning Guide for Cisco UCS M6 Servers - Cisco]
- [Performance Tuning Guide for Cisco UCS M5 Servers White Paper - Cisco]
- [Performance Tuning for Cisco UCS C225 M6 and C245 M6 Rack Servers with 3rd Gen AMD EPYC Processors White Paper - Cisco]

- Products - Performance Tuning for Cisco UCS C125 Rack Server Nodes with AMD Processors (White Paper) - Cisco

Step 7. Click **Create** to create the BIOS Policy.

Procedure 4. Configure Boot Order Policy for M2

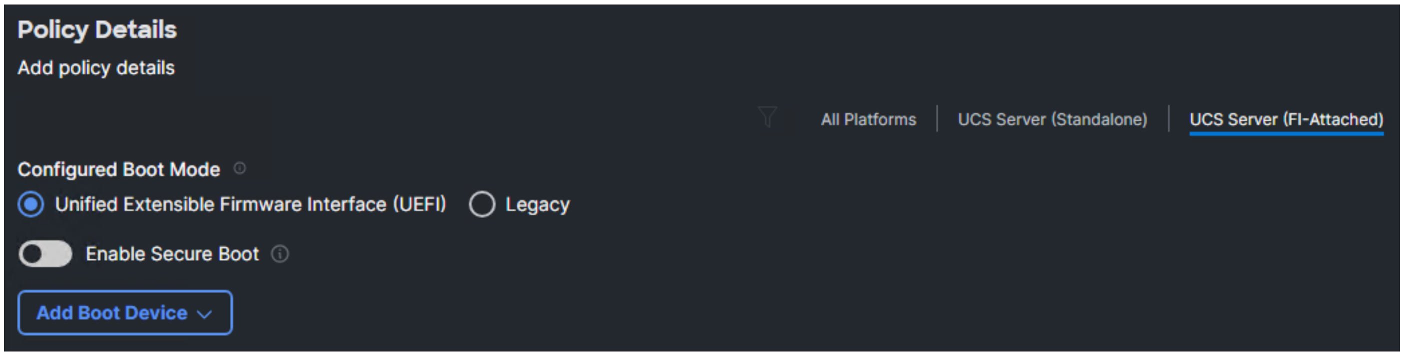
Step 1. Click **Select Policy** next to Boot Order and then, in the pane on the right, click **Create New**.

Step 2. Verify correct organization is selected from the drop-down list (for example, AA02) and provide a name for the policy (for example, AA02-OCP-M2-Boot-Order).

Step 3. Click **Next**.

Step 4. For Configured Boot Mode, select **Unified Extensible Firmware Interface (UEFI)**.

Step 5. Do not turn on Enable Secure Boot.



Configured Boot Mode
[✓ Unified Extensible Firmware Interface (UEFI)] Legacy
Enable Secure Boot: off

Note: It is critical to not enable UEFI Secure Boot. If Secure Boot is enabled, the NVIDIA GPU Operator GPU driver will fail to initialize.

Step 6. Click the **Add Boot Device** drop-down list and select **Virtual Media.**

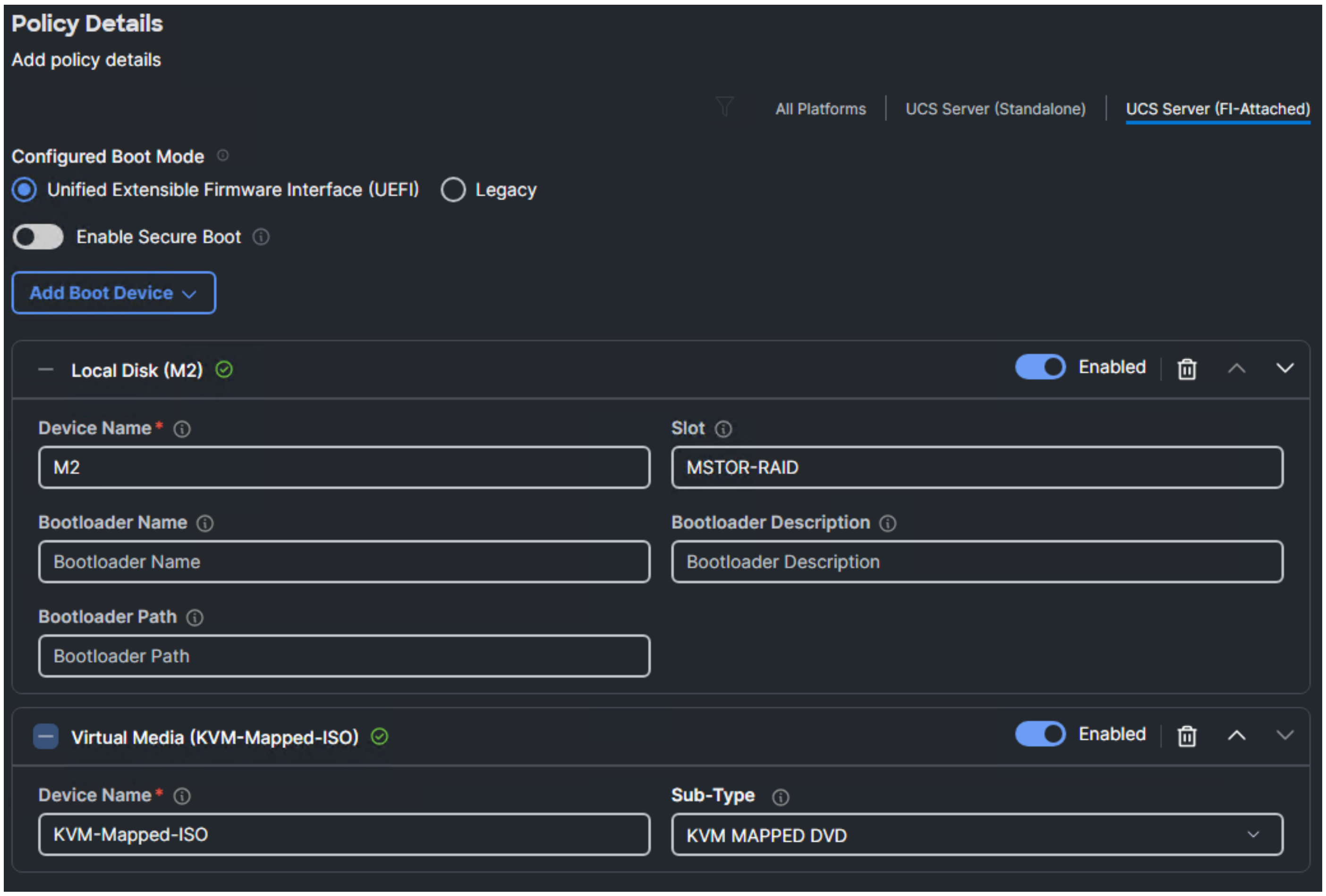
Note: You are entering the Boot Devices in reverse order here to avoid having to move them in the list later.

Step 7. Provide a Device Name (for example, KVM-Mapped-ISO) and then, for the subtype, select **KVM Mapped DVD**.

Step 8. Click the **Add Boot Device** drop-down list and select **Local Disk**.

Step 9. Provide a Device Name (for example, M2) and MSTOR-RAID for the Slot.

Step 10. Verify the order of the boot devices and adjust the boot order as necessary using arrows next to the Delete button.



Policy Details

Add policy details

All Platforms | UCS Server (Standalone) | UCS Server (FI-Attached)

Configured Boot Mode

(●) Unified Extensible Firmware Interface (UEFI) ( ) Legacy

Enable Secure Boot: off

Add Boot Device

Local Disk (M2) — Enabled

Device Name *: M2	Slot: MSTOR-RAID
Bootloader Name: Bootloader Name	Bootloader Description: Bootloader Description
Bootloader Path: Bootloader Path	

Virtual Media (KVM-Mapped-ISO) — Enabled

Device Name *: KVM-Mapped-ISO	Sub-Type: KVM MAPPED DVD

Step 11. Click **Create.**

Procedure 5. Configure Firmware Policy (optional)

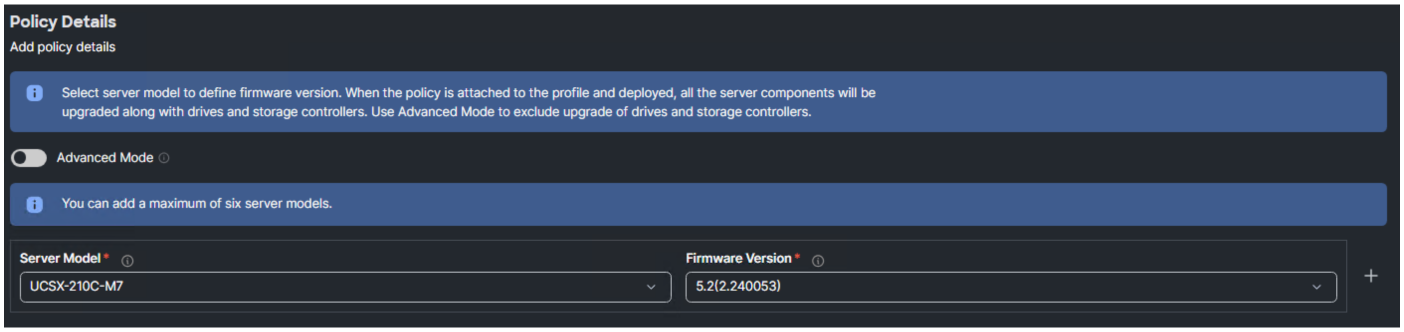
Since Red Hat OCP recommends using homogeneous server types for Masters (and Workers), a Firmware Policy can ensure that all servers are running the appropriate firmware when the Server Profile is deployed.

Step 1. Click **Select Policy** next to Firmware and then, in the pane on the right, click **Create New**.

Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-Firmware). Click **Next**.

Step 3. Select the Server Model (for example, UCSX-210C-M7) and the latest 5.2(2) firmware version.

Step 4. Optionally, other server models can be added using the plus sign.


Policy Details

Add policy details

Select server model to define firmware version. When the policy is attached to the profile and deployed, all the server components will be upgraded along with drives and storage controllers. Use Advanced Mode to exclude upgrade of drives and storage controllers.

Advanced Mode

You can add a maximum of six server models.

Server Model *: UCSX-210C-M7

Firmware Version *: 5.2(2.240053)

Step 5. Click **Create** to create the Firmware Policy.

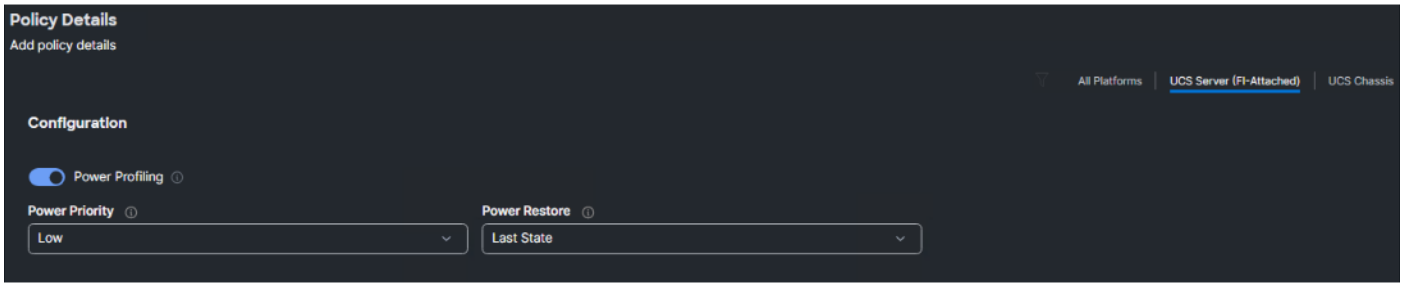
Procedure 6. Configure Power Policy

A Power Policy can be defined and attached to blade servers (Cico UCS X- and B-Series).

Step 1. Click **Select Policy** next to Power and in the pane on the right, click **Create New**.

Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-Server-Power). Click **Next**.

Step 3. Make sure **UCS Server (FI-Attached)** is selected and adjust any of the parameters according to your organizational policies.



Policy Details

Add policy details

Configuration

Power Profiling

Power Priority: Low

Power Restore: Last State

Step 4. Click **Create** to create the Power Policy.

Step 5. Optionally, if you are using Cisco UCS C-Series servers, a Thermal Policy can be created and attached to the profile.

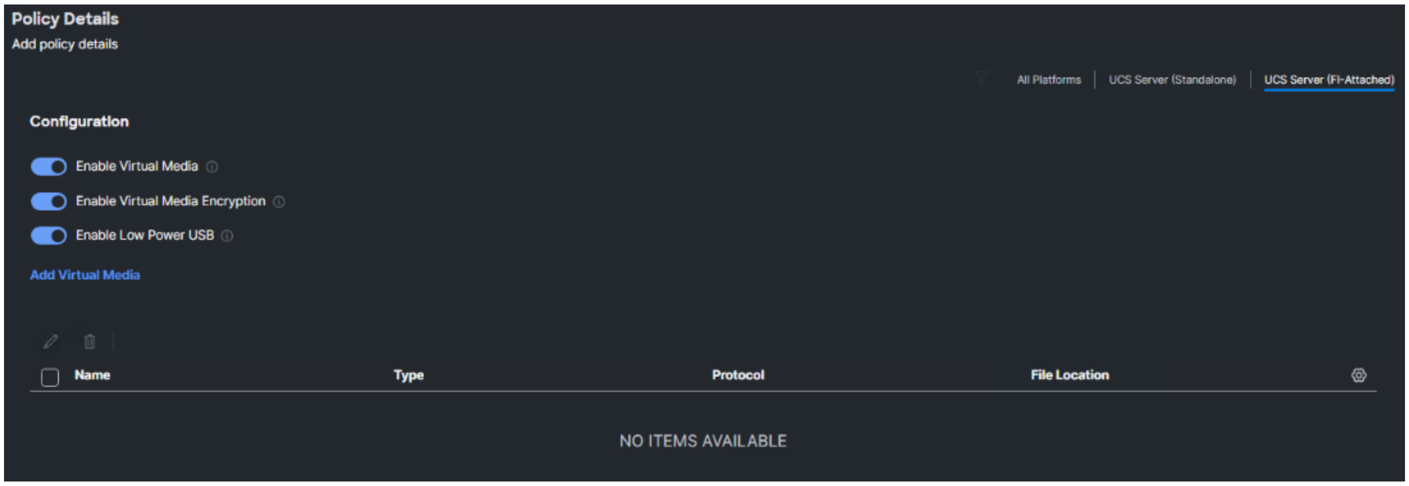
Procedure 7. Configure Virtual Media Policy

Step 1. Click **Select Policy** next to Virtual Media and then, in the pane on the right, click **Create New**.

Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-KVM-Mount-Media). Click **Next**.

Step 3. Ensure that **Enable Virtual Media**, **Enable Virtual Media Encryption**, and **Enable Low Power USB** are turned on.

Step 4. Do not Add Virtual Media at this time, but the policy can be modified and used to map an ISO for a CIMC Mapped DVD.


Policy Details

Add policy details

All Platforms | UCS Server (Standalone) | UCS Server (FI-Attached)

Configuration

Enable Virtual Media

Enable Virtual Media Encryption

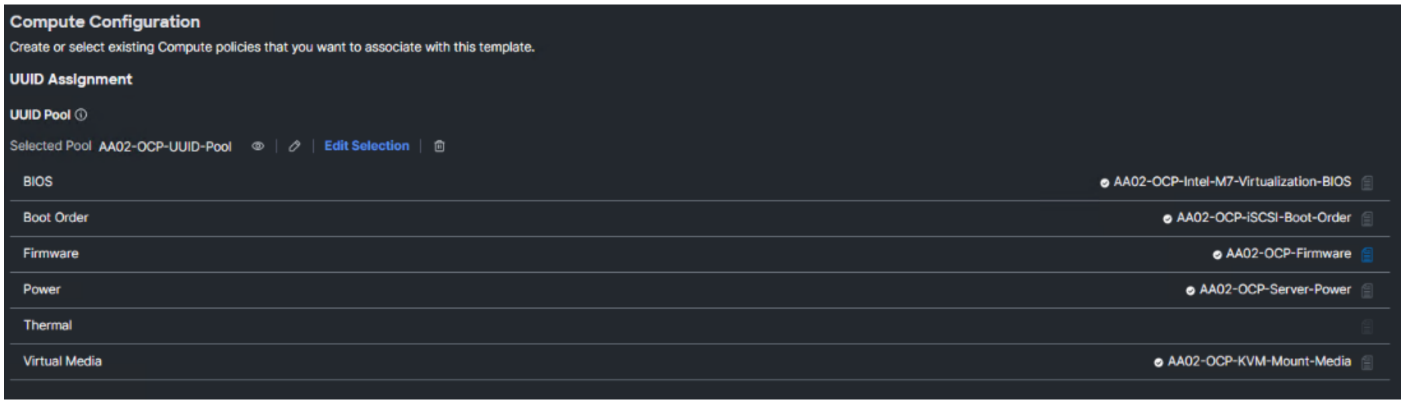
Enable Low Power USB

Add Virtual Media

Name	Type	Protocol	File Location

NO ITEMS AVAILABLE

Step 5. Click **Create** to create the Virtual Media Policy.


Compute Configuration

Create or select existing Compute policies that you want to associate with this template.

UUID Assignment

UUID Pool

Selected Pool AA02-OCP-UUID-Pool | Edit Selection

BIOS	AA02-OCP-Intel-M7-Virtualization-BIOS
Boot Order	AA02-OCP-iSCSI-Boot-Order
Firmware	AA02-OCP-Firmware
Power	AA02-OCP-Server-Power
Thermal	
Virtual Media	AA02-OCP-KVM-Mount-Media

Step 6. Click **Next** to move to Management Configuration.

Management Configuration

The following policies are added to the management configuration:

- IMC Access to define the pool of IP addresses for compute node KVM access
- IPMI Over LAN to allow Intersight to manage IPMI messages
- Local User to provide local administrator to access KVM
- Virtual KVM to allow the Tunneled KVM

Procedure 1. Configure Cisco IMC Access Policy

Step 1. Click **Select Policy** next to IMC Access and then, in the pane on the right, click **Create New**.

Step 2. Verify correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-IMC-Access-Policy).

Step 3. Click **Next**.

Note: Since certain features are not yet enabled for Out-of-Band Configuration (accessed via the Fabric Interconnect mgmt0 ports), we are bringing in the OOB-MGMT VLAN through the Fabric Interconnect Uplinks and mapping it as the In-Band Configuration VLAN.

Step 4. Ensure **UCS Server (FI-Attached)** is selected.

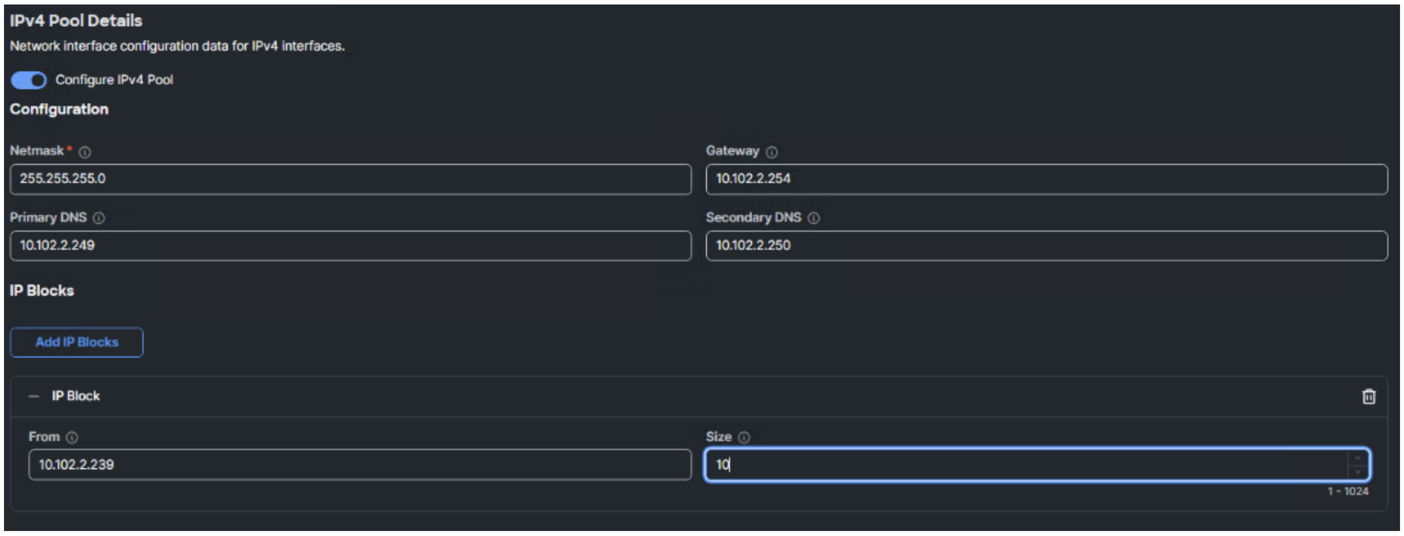
Step 5. **Enable** In-Band Configuration. Enter the OCP-BareMetal VLAN ID (for example, 1022) and select "**IPv4 address configuration**."

Step 6. Under IP Pool, click **Select IP Pool** and then, in the pane on the right, click **Create New.**

Step 7. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-BareMetal-IP-Pool). Click **Next.**

Step 8. Ensure **Configure IPv4 Pool** is selected and provide the information to define a unique pool for KVM IP address assignment including an IP Block (added by clicking **Add IP Blocks**).

Note: You will need the IP addresses of the OCP DNS servers.


IPv4 Pool Details

Network interface configuration data for IPv4 interfaces.

Configure IPv4 Pool

Configuration

Netmask *: 255.255.255.0

Gateway: 10.102.2.254

Primary DNS: 10.102.2.249

Secondary DNS: 10.102.2.250

IP Blocks

IP Block

From: 10.102.2.239

Size: 10

1 - 1024

Note: The management IP pool subnet should be accessible from the host that is trying to open the KVM connection. In the example shown here, the hosts trying to open a KVM connection would need to be able to route to the 10.102.2.0/24 subnet.

Step 9. Click **Next.**

Step 10. Deselect **Configure IPv6 Pool.**

Step 11. Click **Create** to finish configuring the IP address pool.

Step 12. Click **Create** to finish configuring the IMC access policy.

Procedure 2. Configure IPMI Over LAN Policy

The IPMI Over LAN Policy can be used to allow both IPMI and Redfish connectivity to Cisco UCS Servers.

Step 1. Click **Select Policy** next to IPMI Over LAN and then, in the pane on the right, click **Create New.**

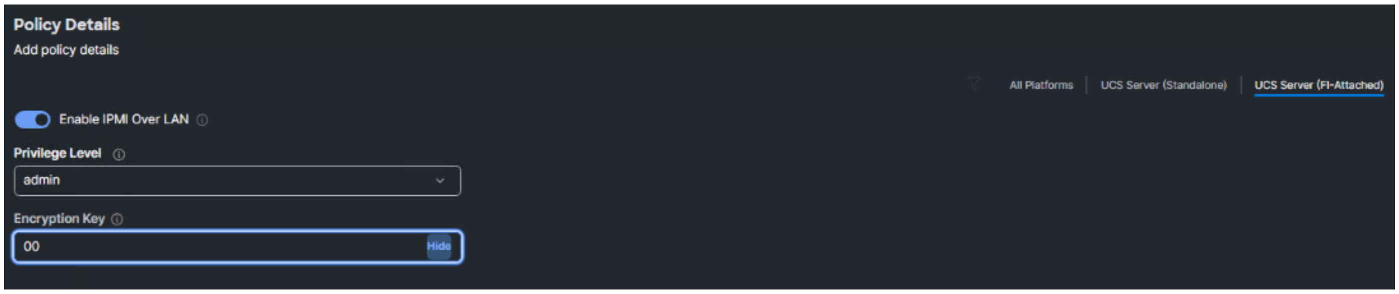
Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02) and provide a name for the policy (for example, AA02-OCP-IPMIoLAN-Policy). Click **Next.**

Step 3. Ensure **UCS Server (FI-Attached)** is selected.

Step 4. Ensure **Enable IPMI Over LAN** is selected.

Step 5. From the **Privilege Level** drop-down list, select **admin**.

Step 6. For Encryption Key, enter **00** to disable encryption.



Enable IPMI Over LAN

Privilege Level: admin

Encryption Key: 00

Step 7. Click **Create** to create the IPMI Over LAN policy.

Procedure 3. Configure Local User Policy

Step 1. Click **Select Policy** next to Local User and then, in the pane on the right, click **Create New**.

Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-LocalUser-Policy). Click **Next**.

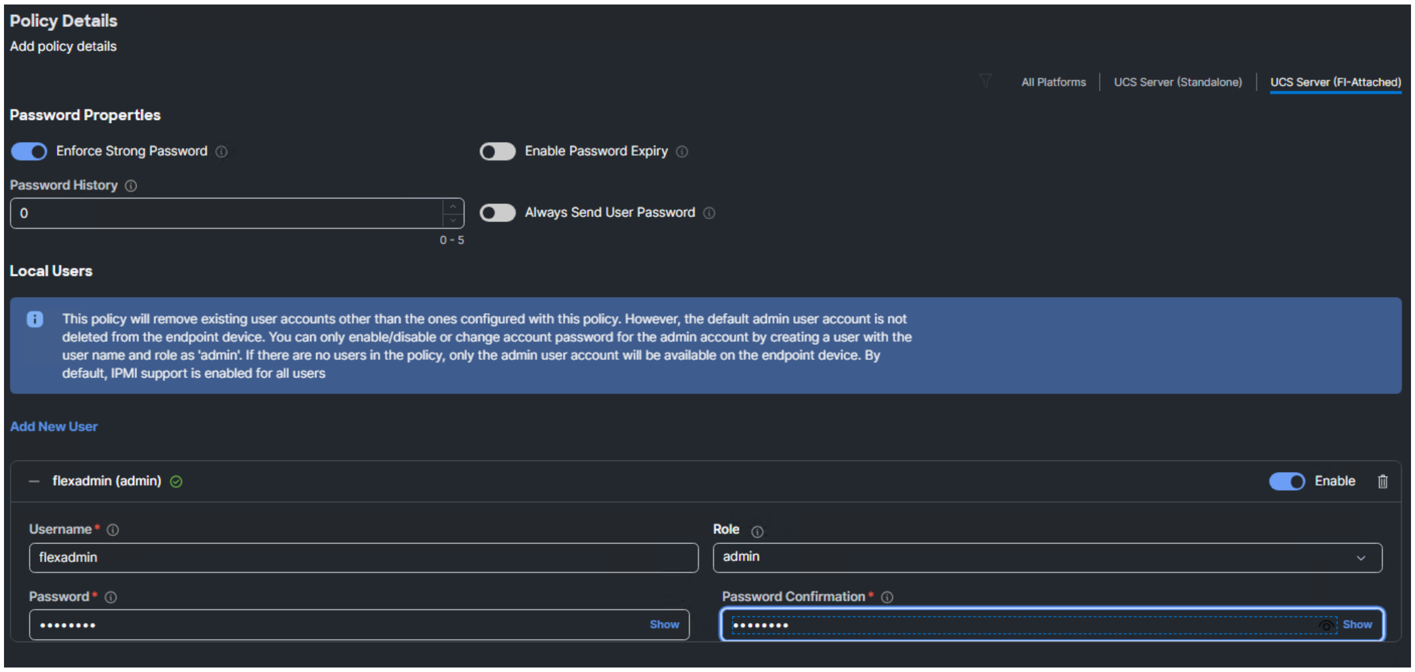
Step 3. Verify that **UCS Server (FI-Attached)** is selected.

Step 4. Verify that **Enforce Strong Password** is selected.

Step 5. Enter **0** under Password History.

Step 6. Click **Add New User**.

Step 7. Provide the username (for example, flexadmin), select a role (for example, admin), and provide a password and password confirmation.


Policy Details

Add policy details

All Platforms | UCS Server (Standalone) | UCS Server (FI-Attached)

Password Properties

Enforce Strong Password: on

Enable Password Expiry: off

Password History: 0 (0 - 5)

Always Send User Password: off

Local Users

This policy will remove existing user accounts other than the ones configured with this policy. However, the default admin user account is not deleted from the endpoint device. You can only enable/disable or change account password for the admin account by creating a user with the user name and role as 'admin'. If there are no users in the policy, only the admin user account will be available on the endpoint device. By default, IPMI support is enabled for all users

Add New User

flexadmin (admin) — Enable

Username*: flexadmin

Role: admin

Password*: ••••••••

Password Confirmation*: ••••••••

Note: The username and password combination defined here will be used as an alternate to log in to KVMs and can be used for IPMI.

Step 8. Click **Create** to complete configuring the Local User policy.

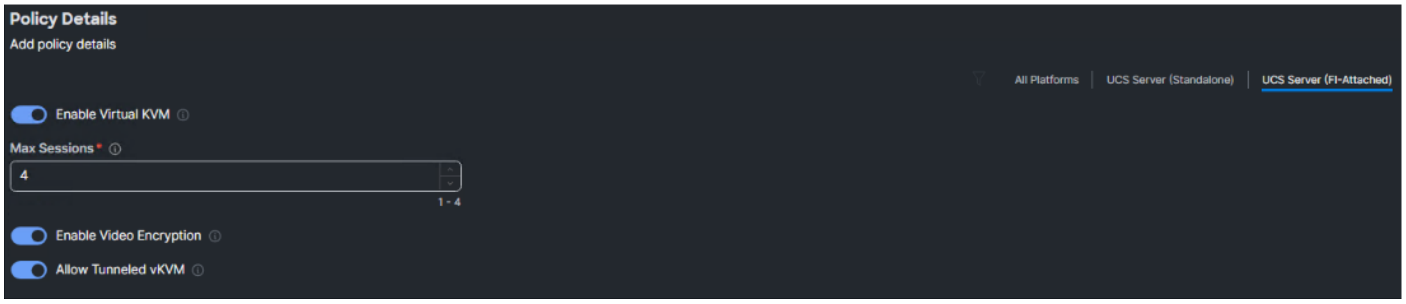
Procedure 4. Configure Virtual KVM Policy

Step 1. Click **Select Policy** next to Virtual KVM and then, in the pane on the right, click **Create New**.

Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-Virtual-KVM). Click **Next**.

Step 3. Verify that **UCS Server (FI-Attached)** is selected.

Step 4. Turn on **Allow Tunneled vKVM**.


Policy Details

Add policy details

All Platforms | UCS Server (Standalone) | UCS Server (FI-Attached)

Enable Virtual KVM: on

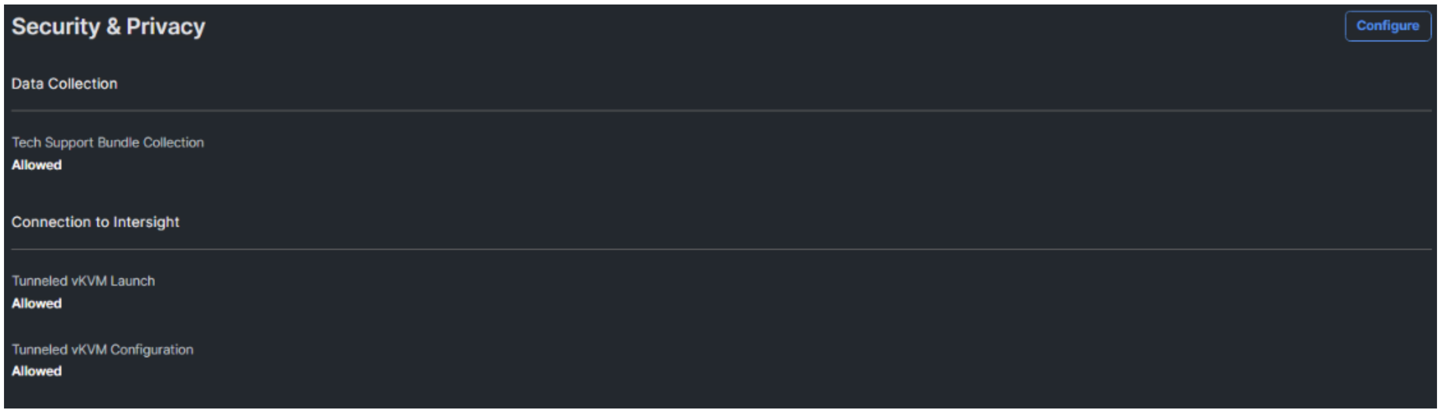
Max Sessions*: 4 (1 - 4)

Enable Video Encryption: on

Allow Tunneled vKVM: on

Step 5. Click **Create**.

Note: To fully enable Tunneled KVM, once the Server Profile Template has been created, go to **System > Settings > Security and Privacy** and click **Configure**. Turn on **Allow Tunneled vKVM Launch** and **Allow Tunneled vKVM Configuration**. If Tunneled vKVM Launch and Tunneled vKVM Configuration are not Allowed, click **Configure** to change these settings.



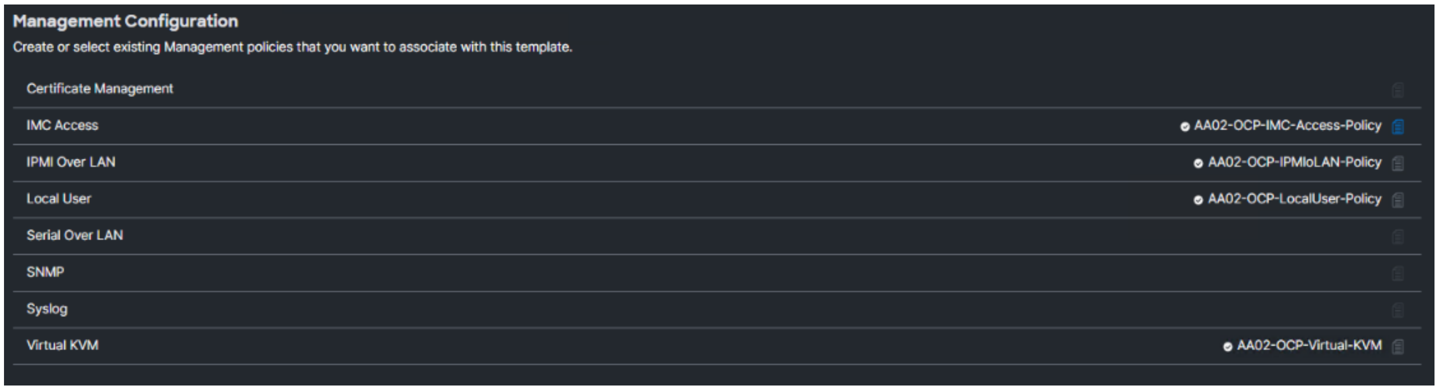
Data Collection

Tech Support Bundle Collection
Allowed

Connection to Intersight

Tunneled vKVM Launch
Allowed

Tunneled vKVM Configuration
Allowed


Management Configuration
Create or select existing Management policies that you want to associate with this template.

Certificate Management	
IMC Access	AA02-OCP-IMC-Access-Policy
IPMI Over LAN	AA02-OCP-IPMIoLAN-Policy
Local User	AA02-OCP-LocalUser-Policy
Serial Over LAN	
SNMP	
Syslog	
Virtual KVM	AA02-OCP-Virtual-KVM

Step 6. Click **Next** to go to Storage Configuration.

Storage Configuration

Procedure 1. Storage Configuration (optional)

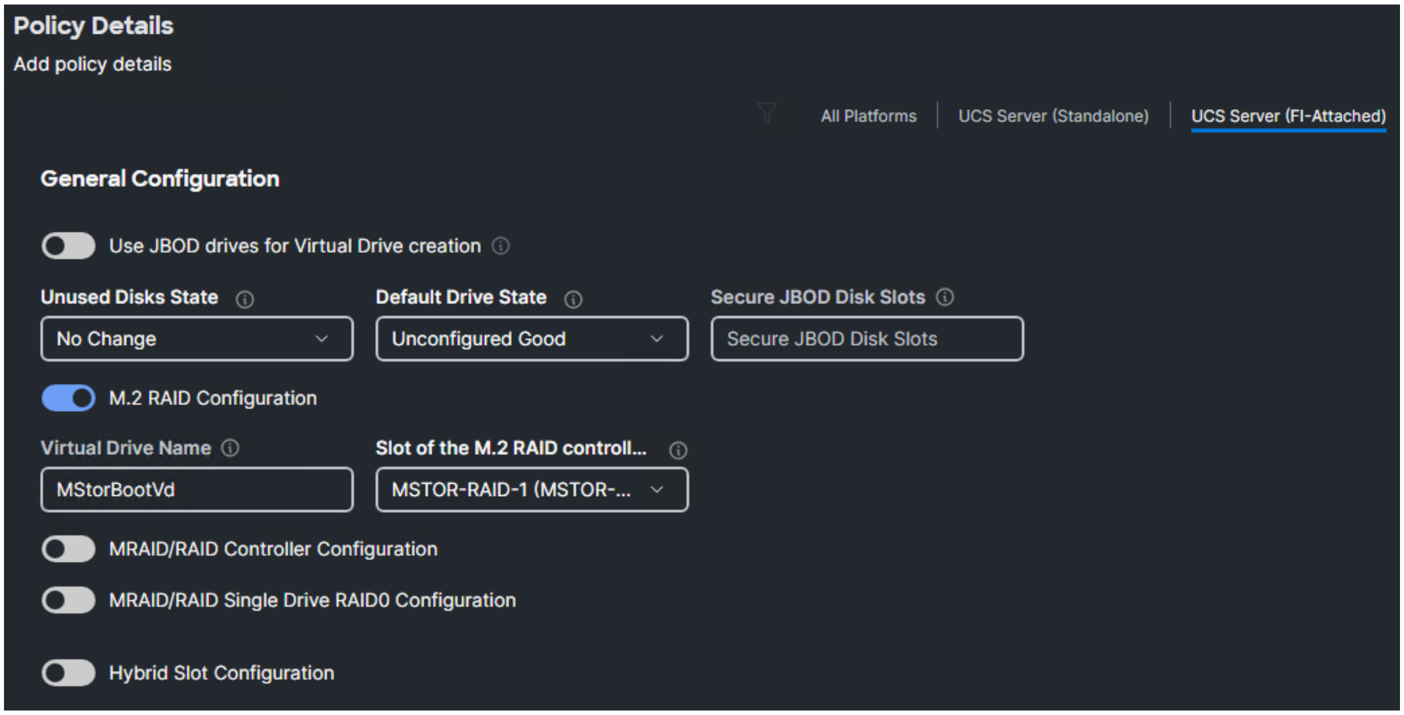
If you have two M.2 drives in your servers you can create an optional policy to mirror these drives using RAID1.

Step 1. If it is not necessary to configure a Storage Policy, click **Next** to continue to Network Configuration.

Step 2. Click **Select Policy** and in the pane on the right-click **Create New**.

Step 3. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-M.2-RAID1-Storage). Click **Next**.

Step 4. Enable **M.2 RAID Configuration** and leave the default Virtual Drive Name and Slot of the M.2 RAID controller field values, or values appropriate to your environment. Click **Create**.



Use JBOD drives for Virtual Drive creation: off

Unused Disks State: No Change

Default Drive State: Unconfigured Good

Secure JBOD Disk Slots: Secure JBOD Disk Slots

M.2 RAID Configuration: on

Virtual Drive Name: MStorBootVd

Slot of the M.2 RAID controll...: MSTOR-RAID-1 (MSTOR-...

MRAID/RAID Controller Configuration: off

MRAID/RAID Single Drive RAID0 Configuration: off

Hybrid Slot Configuration: off

Step 5. Click **Next.**

Network Configuration

Procedure 1. Create Network Configuration – LAN Connectivity for Master Nodes

The LAN connectivity policy defines the connections and network communication resources between the server and the LAN. This policy uses pools to assign MAC addresses to servers and to identify the vNICs that the servers use to communicate with the network. For iSCSI hosts, this policy also defines an IQN address pool.

For consistent vNIC placement, manual vNIC placement is utilized. Additionally, the assumption is being made here that each server contains only one VIC card and Simple placement, which adds vNICs to the first VIC, is being used. If you have more than one VIC in a server, the Advanced placement will need to be used.

The Master hosts use 1 vNIC configured as listed in Table 6.

Table 6. vNIC placement for iSCSI connected storage for OCP Master Nodes

vNIC/vHBA Name	Switch ID	PCI Order	Fabric Failover	Native VLAN	Allowed VLANs	MTU
eno5	A	0	Y	OCP-BareMetal-MGMT	OCP-BareMetal-MGMT	1500

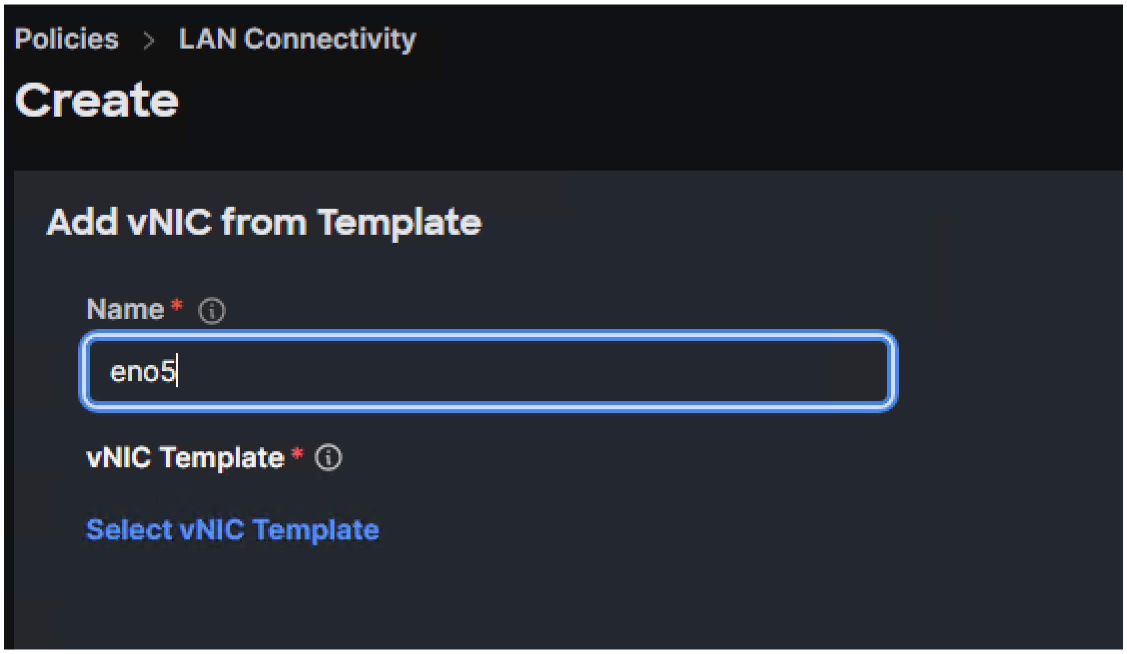
Step 1. Click **Select Policy** next to LAN Connectivity and then, in the pane on the right, click **Create New.**

Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP), provide a name for the policy (for example, AA02-OCP-Master-M2Bt-5G-LANConn) and select **UCS Server (FI-Attached)** under Target Platform. Click **Next.**

Step 3. Leave **None** selected under IQN and under vNIC Configuration, select **Manual vNICs Placement**.

Step 4. Use the Add drop-down list to select **vNIC from Template**.

Step 5. Enter the name for the vNIC from the table above (for example, eno5) and click **Select vNIC Template**.


Policies > LAN Connectivity

Create

Add vNIC from Template

Name * : eno5

vNIC Template *

Step 6. In the upper right, click **Create New**.

Step 7. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the vNIC Template (for example, AA02-OCP-BareMetal-MGMT-vNIC). Click **Next.**

Procedure 2. Create MAC Address Pool for Fabric A and B

Note: When creating the first vNIC, the MAC address pool has not been defined yet, therefore a new MAC address pool will need to be created. Two separate MAC address pools are configured for each Fabric. MAC-Pool-A will be used for all Fabric-A vNICs, and MAC-Pool-B will be used for all Fabric-B vNICs. Adjust the values in the table for your environment.

Table 7. MAC Address Pools

Pool Name	Starting MAC Address	Size	vNICs
MAC-Pool-A	00:25:B5:A2:0A:00	64*	eno5, eno6
MAC-Pool-B	00:25:B5:A2:0B:00	64*	eno7, eno8

Note: For Masters, each server requires 1 MAC address from MAC-Pool-A, and for Workers, each server requires 3 MAC addresses from MAC-Pool-A and 2 MAC addresses from MAC-Pool-B. Adjust the size of the pool according to your requirements.

Step 1. Click **Select Pool** under MAC Pool and then, in the pane on the right, click **Create New**.

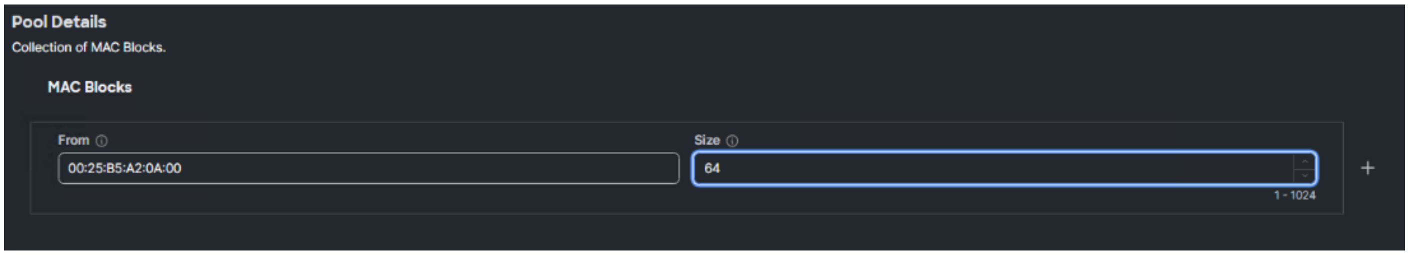
Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the pool from Table 7 with the prefix applied depending on the vNIC being created (for example, AA02-OCP-MAC-Pool-A for Fabric A).

Step 3. Click **Next**.

Step 4. Provide the starting MAC address from Table 7 (for example, 00:25:B5:A2:0A:00)

Step 5. For ease of troubleshooting FlexPod, some additional information is always coded into the MAC address pool. For example, in the starting address 00:25:B5:A2:0A:00, A2 is the rack ID and 0A indicates Fabric A.

Step 6. Provide the size of the MAC address pool from Table 7 (for example, 64).



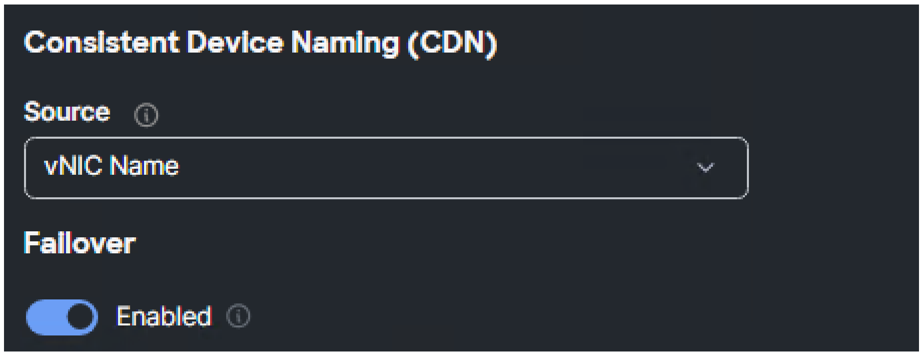
From: 00:25:B5:A2:0A:00
Size: 64

Step 7. Click **Create** to finish creating the MAC address pool.

Step 8. From the Create vNIC Template window, provide the Switch ID from Table 6.

Step 9. For Consistent Device Naming (CDN), from the drop-down list, select **vNIC Name**.

Step 10. For Failover, set the value from Table 6.



Source: vNIC Name
Failover: Enabled

Procedure 3. Create Ethernet Network Group Policy

Ethernet Network Group policies will be created and reused on applicable vNICs as explained below. The ethernet network group policy defines the VLANs allowed for a particular vNIC, therefore multiple network group policies will be defined for this deployment as listed in Table 8.

Table 8. Ethernet Group Policy Values

Group Policy Name	Native VLAN	Apply to vNICs	Allowed VLANs
AA02-OCP-BareMetal-NetGrp	OCP-BareMetal-MGMT (1022)	eno5	OCP-BareMetal-MGMT

Group Policy Name	Native VLAN	Apply to vNICs	Allowed VLANs
AA02-OCP-iSCSI-NVMe-TCP-A-NetGrp	OCP-iSCSI-A (3012)	eno6	-iSCSI-A, OCP-NVMe-TCP-A*
AA02-OCP-iSCSI-NVMe-TCP-B-NetGrp	OCP-iSCSI-B (3022)	eno7	-iSCSI-B, OCP-NVMe-TCP-B*
AA02-OCP-NFS-NetGrp	OCP-NFS	eno8	OCP-NFS

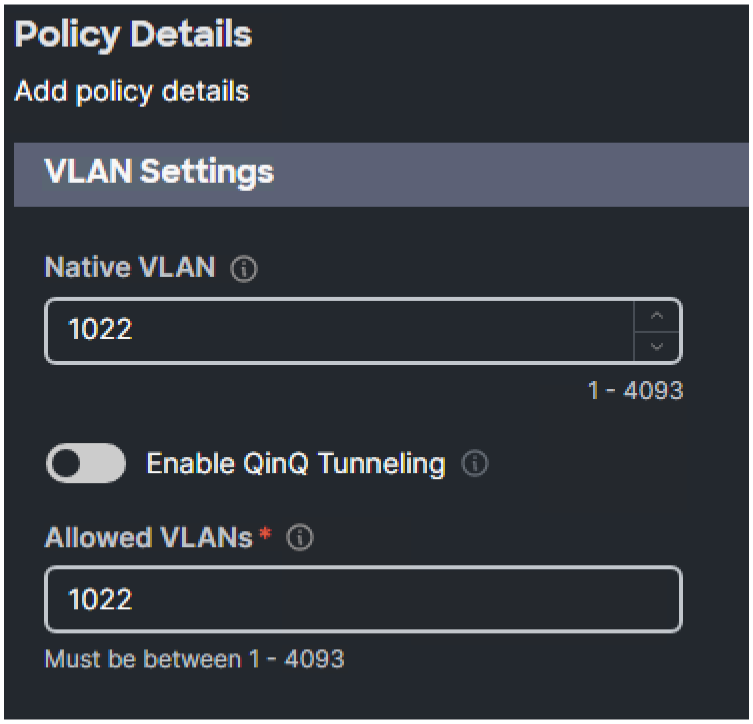
Note: *Add the NVMe-TCP VLANs when using NVMe-TCP.

Step 1. Click **Select Policy** under Ethernet Network Group Policy and then, in the pane on the right, click **Create New**.

Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy from the Table 8 (for example, AA02-OCP-BareMetal-NetGrp).

Step 3. Click **Next**.

Step 4. Enter the Native VLAN ID (for example,1022) and the allowed VLANs (for example, 1022) from Table 2.


Policy Details
Add policy details
VLAN Settings
Native VLAN: 1022
1 - 4093
Enable QinQ Tunneling
Allowed VLANs *: 1022
Must be between 1 - 4093

Step 5. Click **Create** to finish configuring the Ethernet network group policy.

Note: When ethernet group policies are shared between two vNICs, the ethernet group policy only needs to be defined for the first vNIC. For subsequent vNIC policy mapping, click **Select Policy** and pick the previously defined ethernet network group policy from the list.

Procedure 4. Create Ethernet Network Control Policy

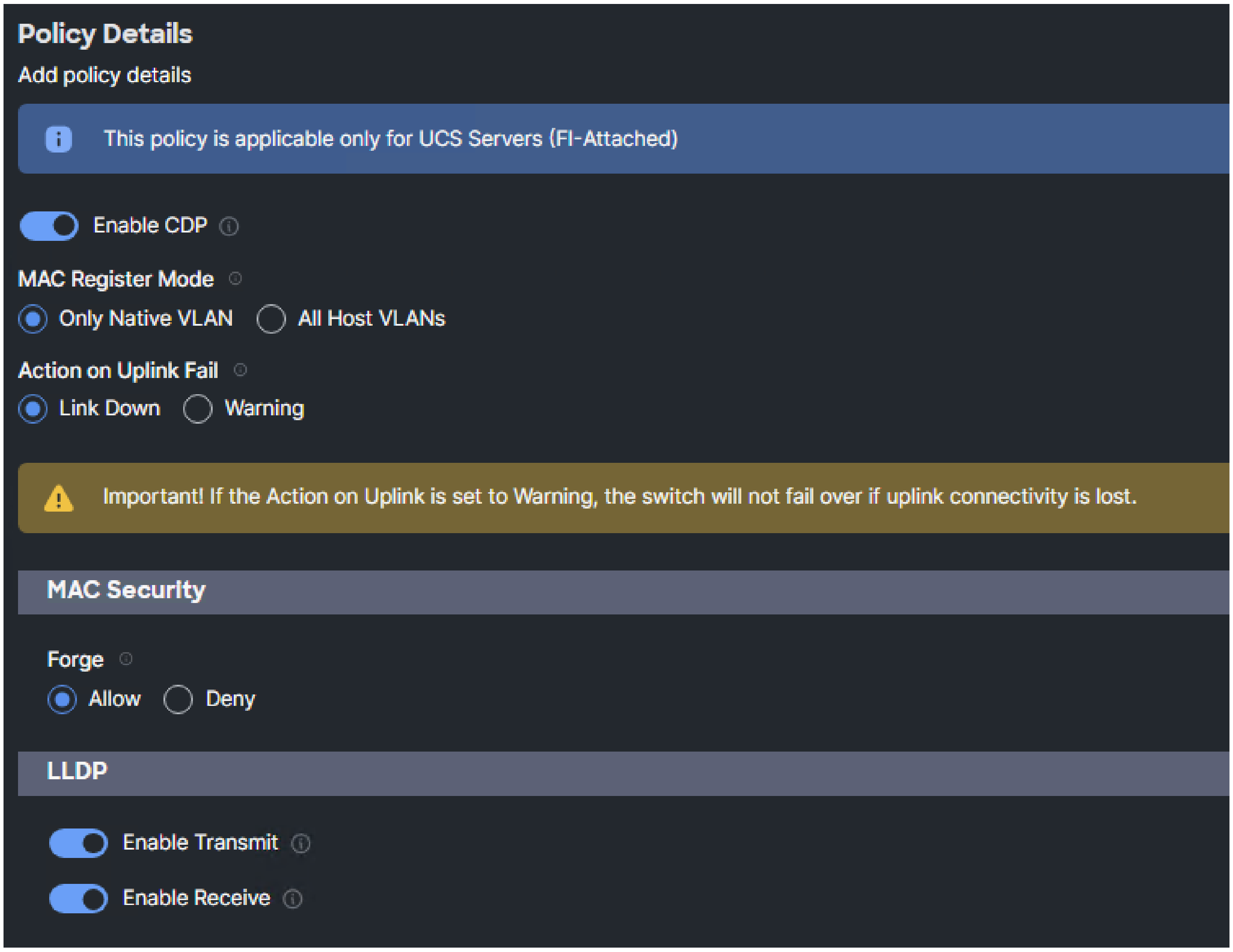
The Ethernet Network Control Policy is used to enable Cisco Discovery Protocol (CDP) and Link Layer Discovery Protocol (LLDP) for the vNICs. A single policy will be created here and reused for all the vNICs.

Step 1. Click **Select Policy** under Ethernet Network Control Policy and then, in the pane on the right, click **Create New**.

Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-Enable-CDP-LLDP).

Step 3. Click **Next**.

Step 4. Enable **Cisco Discovery Protocol (CDP)** and **Enable Transmit** and **Enable Receive** under LLDP.


Policy Details

Add policy details

This policy is applicable only for UCS Servers (FI-Attached)

Enable CDP: enabled

MAC Register Mode: [✓ Only Native VLAN] All Host VLANs

Action on Uplink Fail: [✓ Link Down] Warning

Important! If the Action on Uplink is set to Warning, the switch will not fail over if uplink connectivity is lost.

MAC Security

Forge: [✓ Allow] Deny

LLDP

Enable Transmit: enabled

Enable Receive: enabled

Step 5. Click **Create** to finish creating Ethernet network control policy.

Procedure 5. Create Ethernet QoS Policy

Note: The Ethernet QoS policy is used to enable the appropriate maximum transmission unit (MTU) for all the vNICs. Across the vNICs, two policies will be created (one for MTU 1500 and one for MTU 9000) and reused for all the vNICs.

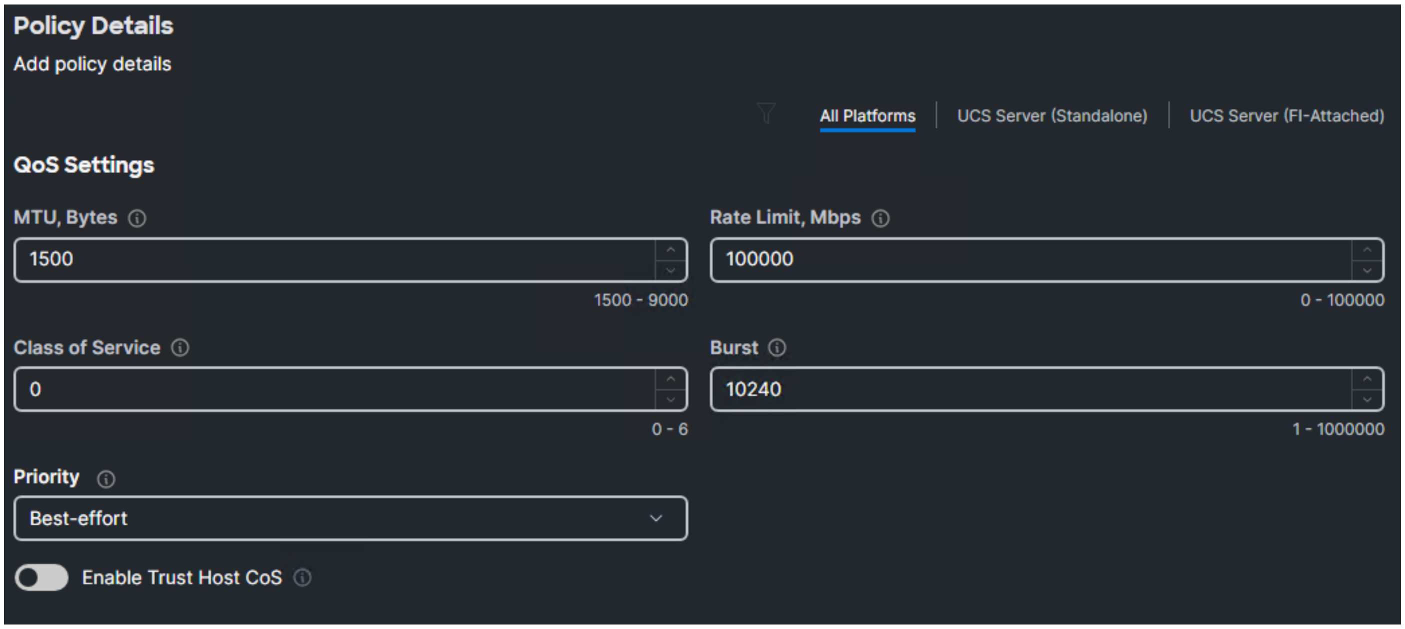
Step 1. Click **Select Policy** under Ethernet QoS and in the pane on the right, click **Create New**.

Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-MTU1500-EthernetQoS). The name of the policy should conform to the MTU from Table 8.

Step 3. Click **Next**.

Step 4. Change the MTU, Bytes value to the value from Table 8.

Step 5. Set the Rate Limit Mbps to **100000**.



MTU, Bytes: 1500 (1500 - 9000)
Rate Limit, Mbps: 100000 (0 - 100000)
Class of Service: 0 (0 - 6)
Burst: 10240 (1 - 1000000)
Priority: Best-effort
Enable Trust Host CoS

Step 6. Click **Create** to finish setting up the Ethernet QoS policy.

Procedure 6. Create Ethernet Adapter Policy

The ethernet adapter policy is used to set the interrupts, send, and receive queues, and queue ring size. The values are set according to the best-practices guidance for the operating system in use. Cisco Intersight provides a default Linux Ethernet Adapter policy for typical Linux deployments.

You can optionally configure a tweaked ethernet adapter policy for additional hardware receive queues handled by multiple CPUs in scenarios where there is a lot of traffic and multiple flows. In this deployment, a modified ethernet adapter policy, AA02-EthAdapter-16RXQs-5G, is created and attached to storage vNICs. Non-storage vNICs will use the default Linux-v2 Ethernet Adapter policy.

Table 9. Ethernet Adapter Policy association to vNICs

Policy Name	vNICs

Policy Name	vNICs
AA02-OCP-EthAdapter-Linux-v2	eno5
AA02-OCP-EthAdapter-16RXQs-5G	eno6, eno7, eno8

Step 1. Click **Select Policy** under Ethernet Adapter and then, in the pane on the right, click **Create New**.

Step 2. Verify the correct organization is selected from the drop-down list (for example, AA02-OCP) and provide a name for the policy (for example, AA02-OCP-EthAdapter-Linux-v2).

Step 3. Click **Select Cisco Provided Configuration** under Cisco Provided Ethernet Adapter Configuration.

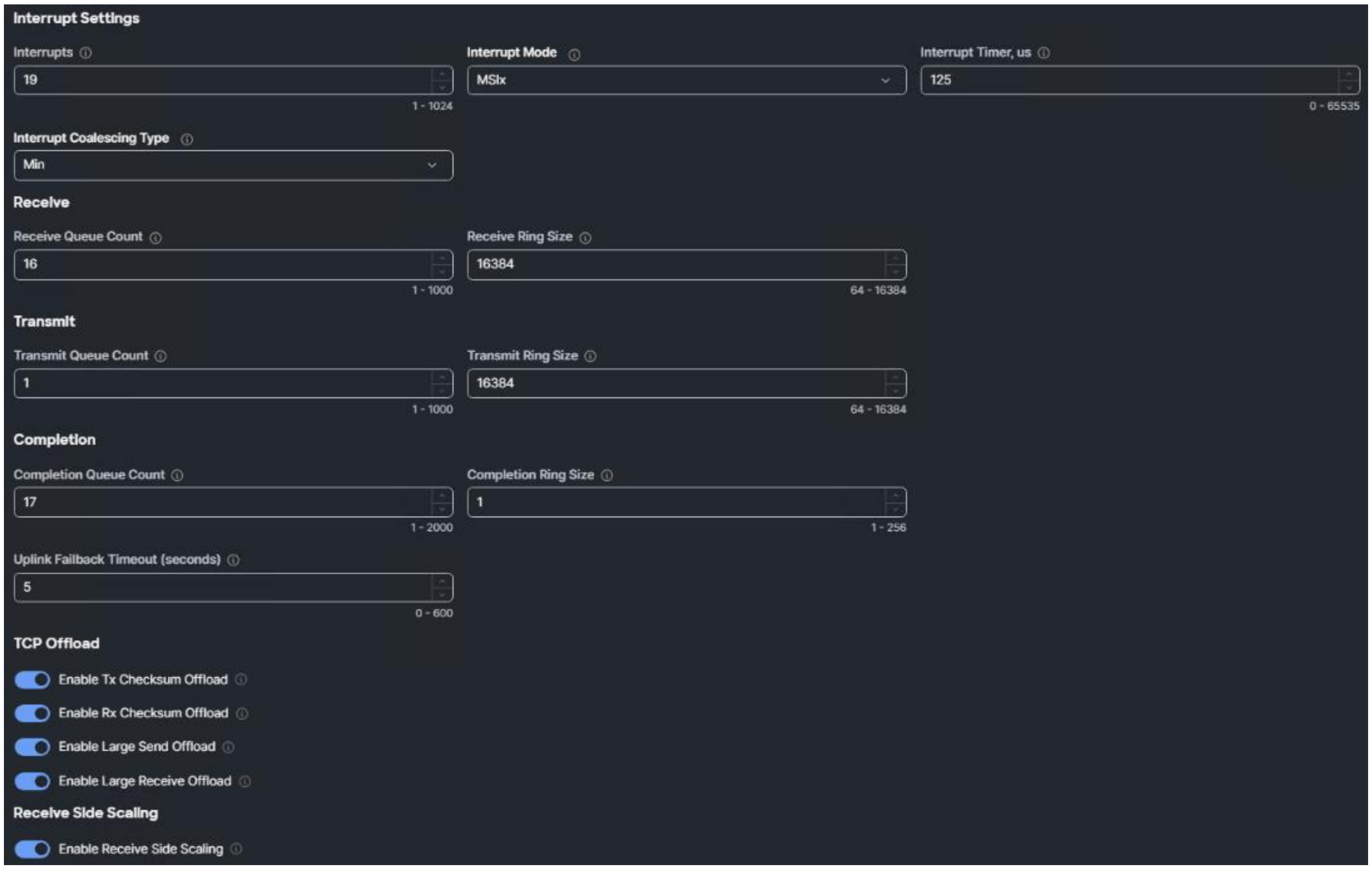
Step 4. From the list, select **Linux-v2**.

Step 5. Click **Next**.

Step 6. For the AA02-OCP-EthAdapter-Linux-v2 policy, click **Create** and skip the rest of the steps in this "Create Ethernet Adapter Policy" section.

Step 7. For the AA02-OCP-EthAdapter-16RXQs-5G policy, make the following modifications to the policy:

- Increase Interrupts to 19
- Increase Receive Queue Count to 16
- Increase Receive Ring Size to 16384 (Leave at 4096 for 4G VICs)
- Increase Transmit Ring Size to 16384 (Leave at 4096 for 4G VICs)
- Increase Completion Queue Count to 17
- Ensure Receive Side Scaling is enabled



Interrupt Settings

Interrupts: 19 (1 - 1024)

Interrupt Mode: MSIx

Interrupt Timer, us: 125 (0 - 65535)

Interrupt Coalescing Type: Min

Receive

Receive Queue Count: 16 (1 - 1000)

Receive Ring Size: 16384 (64 - 16384)

Transmit

Transmit Queue Count: 1 (1 - 1000)

Transmit Ring Size: 16384 (64 - 16384)

Completion

Completion Queue Count: 17 (1 - 2000)

Completion Ring Size: 1 (1 - 256)

Uplink Failback Timeout (seconds): 5 (0 - 600)

TCP Offload

Enable Tx Checksum Offload

Enable Rx Checksum Offload

Enable Large Send Offload

Enable Large Receive Offload

Receive Side Scaling

Enable Receive Side Scaling

Step 8. Click **Create.**

Procedure 7. Add vNIC(s) to LAN Connectivity Policy

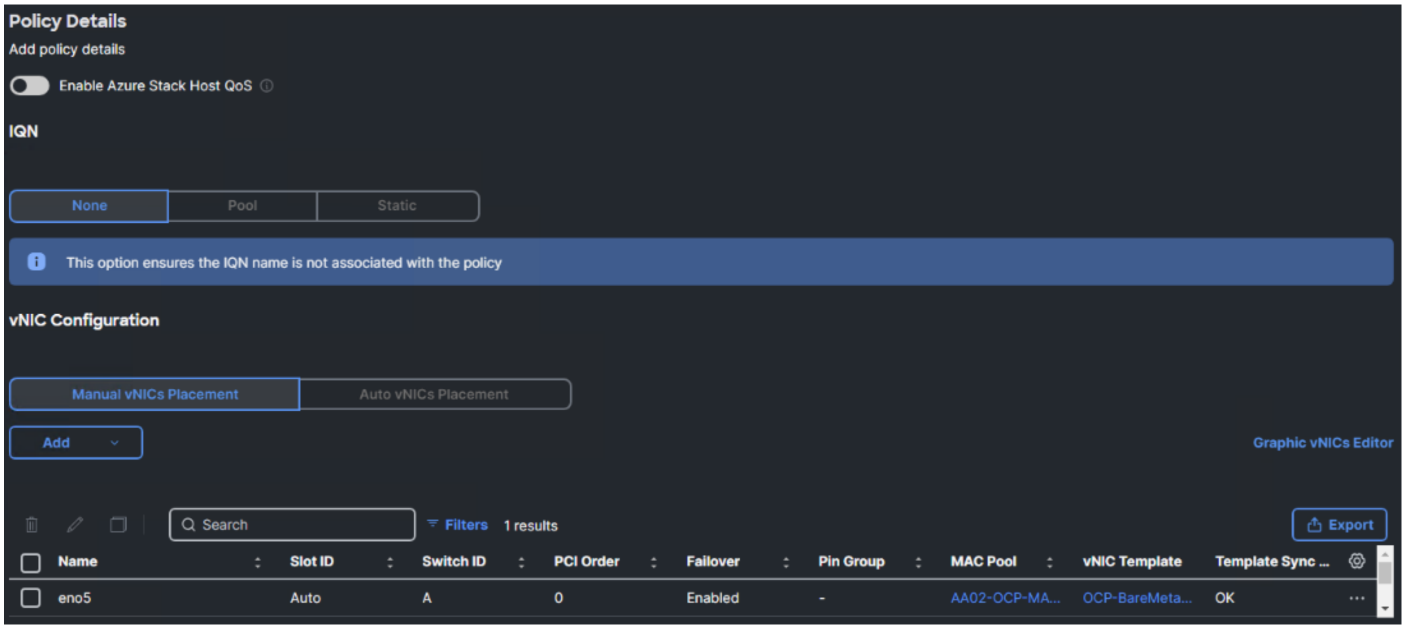
The vNIC Template exists and all policies attached.

Step 1. For PCI Order enter the number from Table 8. Verify the other values.

Step 2. Click **Add** to add the vNIC to the LAN Connectivity Policy.

Step 3. If building the Worker LAN Connectivity Policy, go back to Procedure 1 Create Network Configuration - LAN Connectivity for Master Nodes, Step 4 and repeat the vNIC Template and vNIC creation for all five vNICs often selecting existing policies instead of creating them.

Step 4. Verify all vNICs were successfully created.


Policy Details

Add policy details

Enable Azure Stack Host QoS

IQN

[✓ None] Pool Static

This option ensures the IQN name is not associated with the policy

vNIC Configuration

[✓ Manual vNICs Placement] Auto vNICs Placement

Add

Graphic vNICs Editor

Search Filters 1 results Export

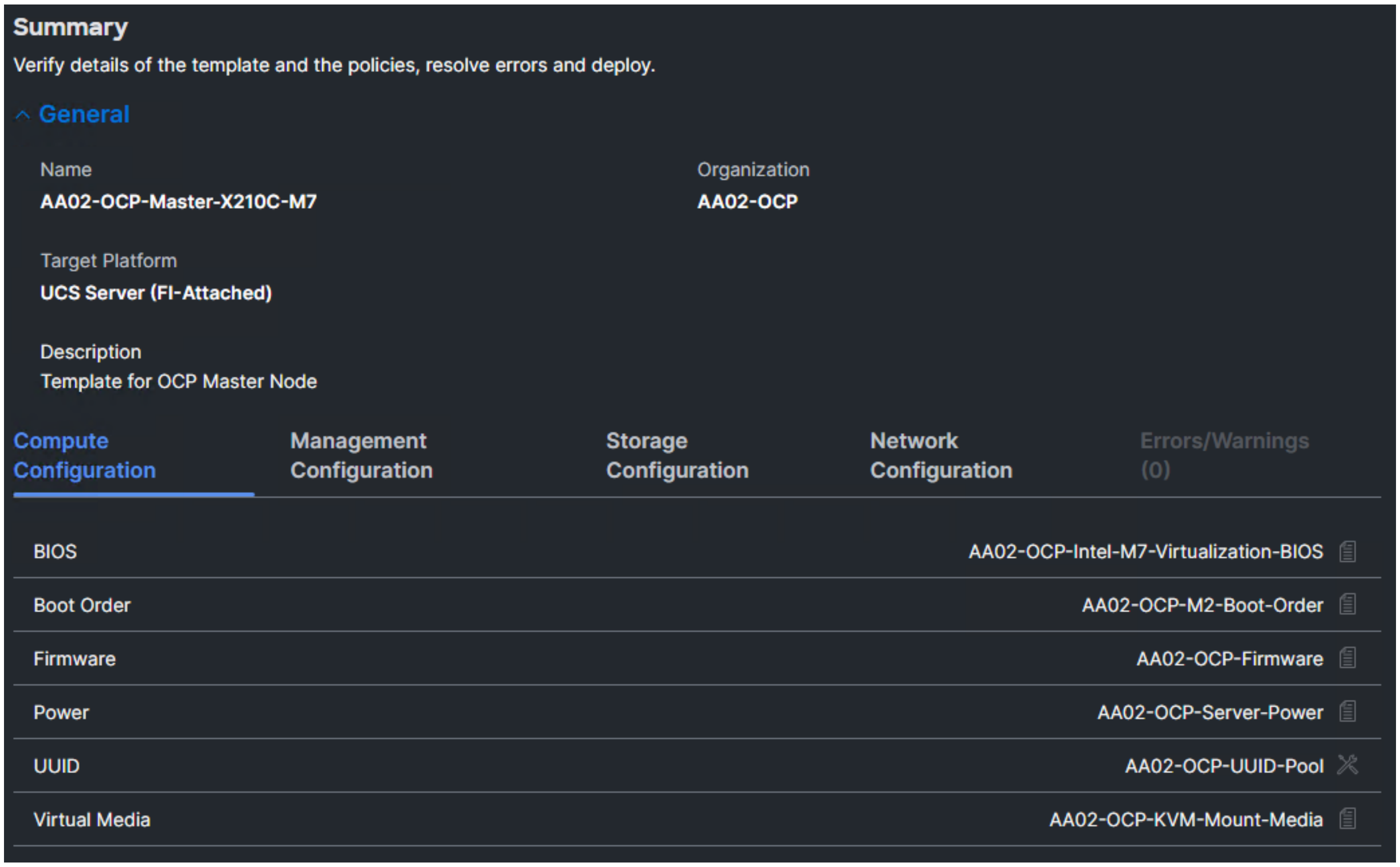
Name	Slot ID	Switch ID	PCI Order	Failover	Pin Group	MAC Pool	vNIC Template	Template Sync ...
eno5	Auto	A	0	Enabled	-	AA02-OCP-MA...	OCP-BareMeta...	OK

Step 5. Click **Create** to finish creating the LAN Connectivity policy.

Procedure 8. Complete the Master Server Profile Template

Step 1. When the LAN connectivity policy is created, click **Next** to move to the Summary screen.

Step 2. On the Summary screen, verify the policies are mapped to various settings. The screenshots below provide the summary view for the OCP Master M.2 Boot server profile template.



Summary

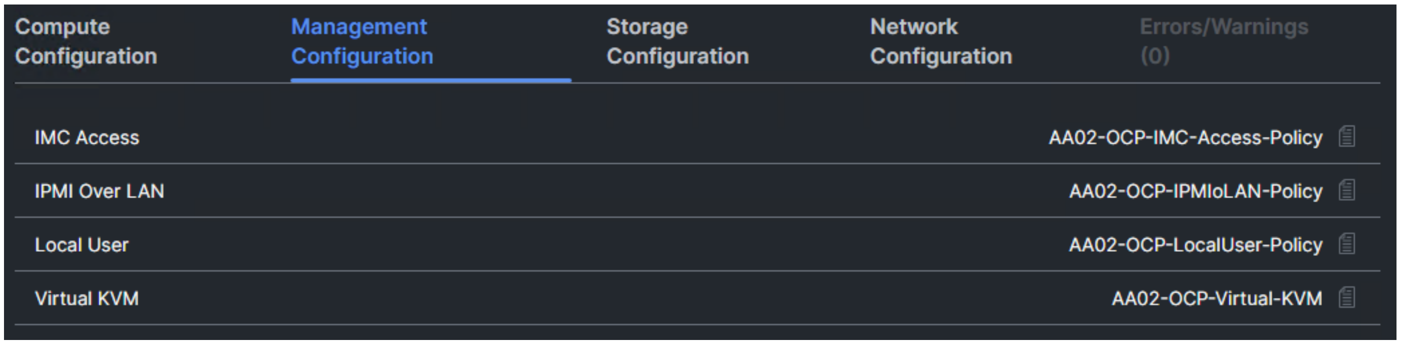
Verify details of the template and the policies, resolve errors and deploy.

General

Name	Organization
AA02-OCP-Master-X210C-M7	AA02-OCP
Target Platform	
UCS Server (FI-Attached)	
Description	
Template for OCP Master Node	

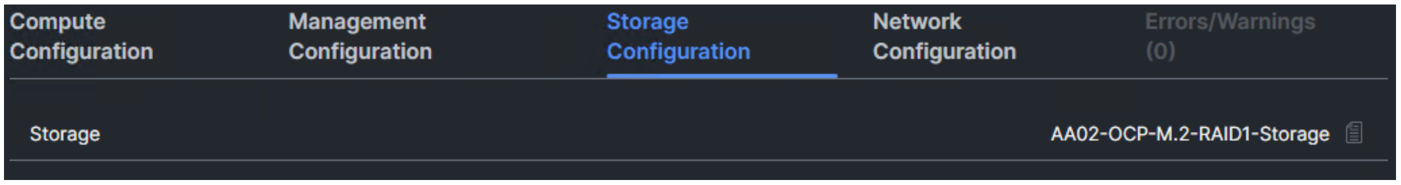
Compute Configuration | Management Configuration | Storage Configuration | Network Configuration | Errors/Warnings (0)

BIOS	AA02-OCP-Intel-M7-Virtualization-BIOS
Boot Order	AA02-OCP-M2-Boot-Order
Firmware	AA02-OCP-Firmware
Power	AA02-OCP-Server-Power
UUID	AA02-OCP-UUID-Pool
Virtual Media	AA02-OCP-KVM-Mount-Media



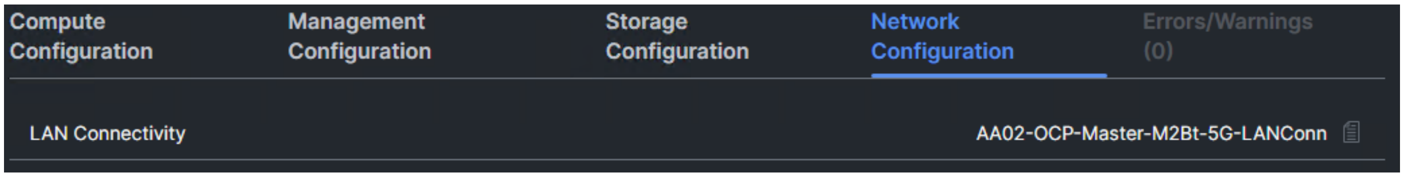
Compute Configuration | Management Configuration | Storage Configuration | Network Configuration | Errors/Warnings (0)

IMC Access	AA02-OCP-IMC-Access-Policy
IPMI Over LAN	AA02-OCP-IPMIoLAN-Policy
Local User	AA02-OCP-LocalUser-Policy
Virtual KVM	AA02-OCP-Virtual-KVM



Compute Configuration | Management Configuration | Storage Configuration | Network Configuration | Errors/Warnings (0)

Storage	AA02-OCP-M.2-RAID1-Storage



Compute Configuration | Management Configuration | Storage Configuration | Network Configuration | Errors/Warnings (0)

LAN Connectivity	AA02-OCP-Master-M2Bt-5G-LANConn

Step 3. Click **Close** to close the template.

Procedure 9. Build the OCP Worker LAN Connectivity Policy

The OCP Master LAN Connectivity Policy can be cloned and four vNICs added to build the OCP Worker LAN Connectivity Policy that will then be used in the OCP Worker Server Profile Template. Table 10 lists the vNICs that will be added to the cloned policy.

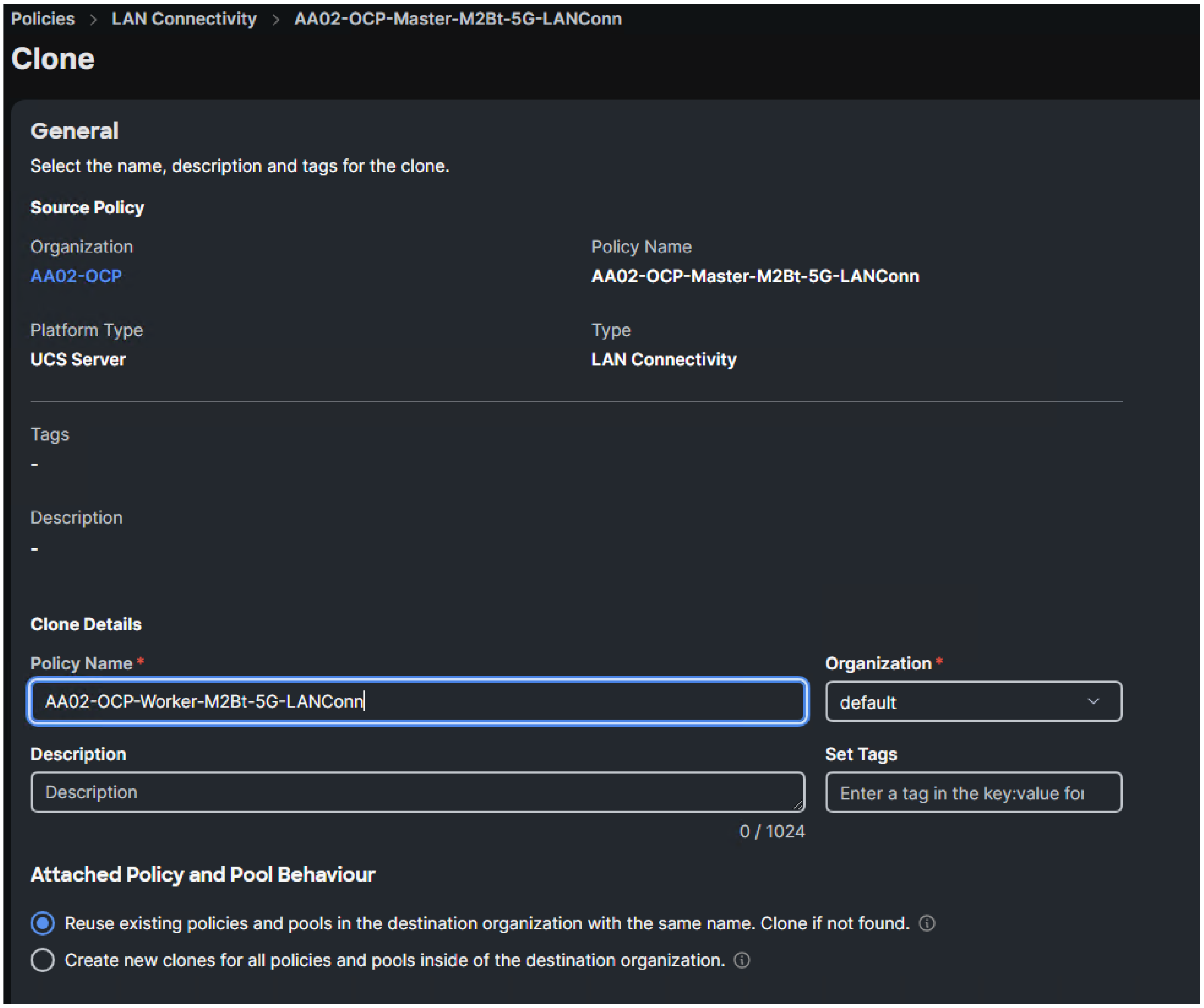
Table 10. vNIC additions for iSCSI connected storage for OCP Worker Nodes

vNIC/vHBA Name	Switch ID	PCI Order	Fabric Failover	Native VLAN	Allowed VLANs	MTU
eno6	A	1	N	OCP-iSCSI-A	OCP-iSCSI-A, OCP-NVMe-TCP-A	9000
eno7	B	2	N	OCP-iSCSI-B	OCP-iSCSI-B, OCP-NVMe-TCP-B	9000
eno8	A	3	N	OCP-NFS	OCP-NFS	9000

Step 1. Log into **Cisco Intersight** and select **Infrastructure Service > Policies**.

Step 2. In the policy list, look for the <org-name>-Master-M2Bt-5G-LANConn or the LAN Connectivity policy created above. Click **...** to the right of the policy and select **Clone**.

Step 3. Change the name of the cloned policy to something like AA02-OCP-Worker-M2Bt-5G-LANConn and select the correct Organization (for example, AA02-OCP).


Policies > LAN Connectivity > AA02-OCP-Master-M2Bt-5G-LANConn

Clone

General

Select the name, description and tags for the clone.

Source Policy

Organization: AA02-OCP	Policy Name: AA02-OCP-Master-M2Bt-5G-LANConn
Platform Type: UCS Server	Type: LAN Connectivity

Tags: -

Description: -

Clone Details

Policy Name *: AA02-OCP-Worker-M2Bt-5G-LANConn

Organization *: default

Description: Description (0 / 1024)

Set Tags: Enter a tag in the key:value for

Attached Policy and Pool Behaviour

[✓ Reuse existing policies and pools in the destination organization with the same name. Clone if not found.]

Create new clones for all policies and pools inside of the destination organization.

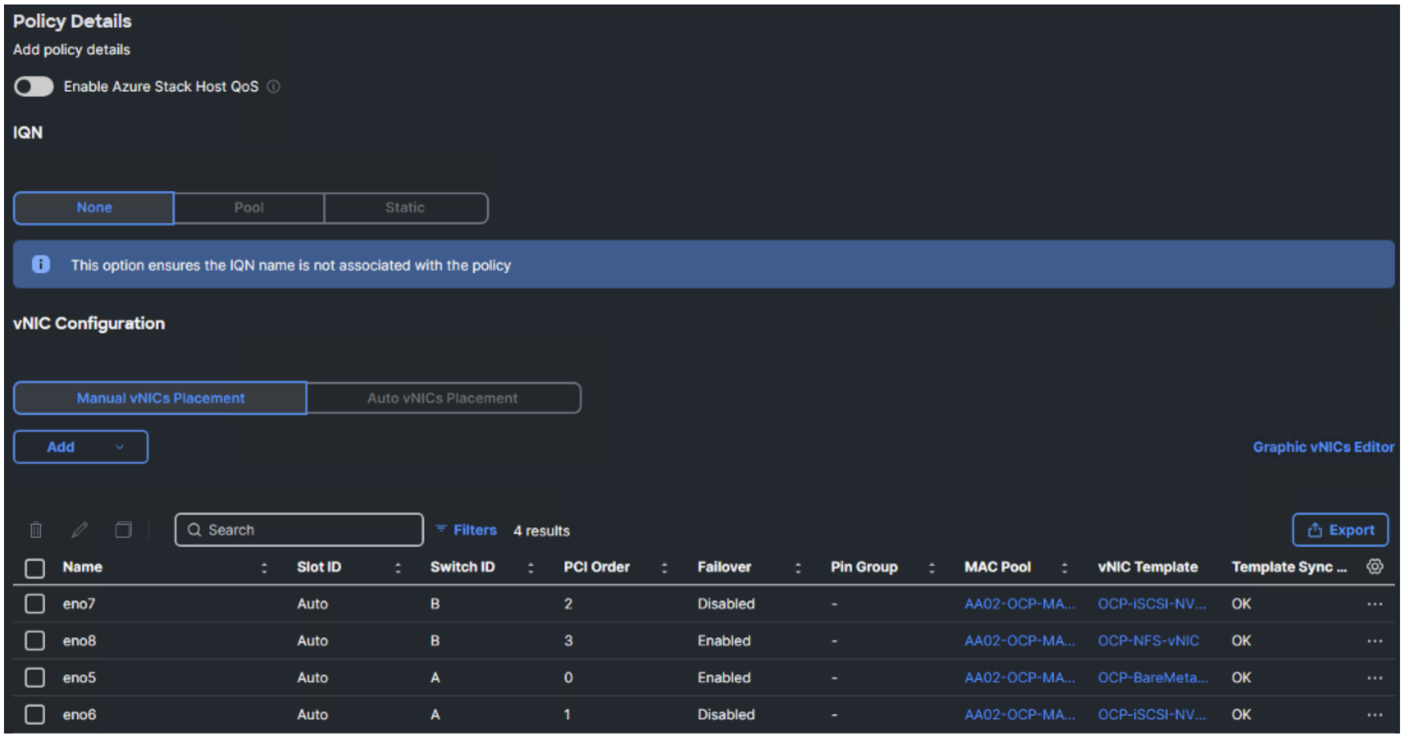
Step 4. Click **Clone** to clone the policy.

Step 5. From the Policies window, click the refresh button to refresh the list. The newly cloned policy should now appear at the top of the list. Click **...** to the right of the newly cloned policy and select **Edit**.

Step 6. Click **Next**.

Step 7. Go to Procedure 1 Create Network Configuration - LAN Connectivity for Master Nodes and start at Step 4 to add the 4 vNICs from Template listed in Table 10.

The OCP-Worker-M2Bt-5G-LANConn policy is now built and can be added to the OCP Worker Server Profile Template.


Policy Details

Add policy details

Enable Azure Stack Host QoS

IQN

None | Pool | Static

This option ensures the IQN name is not associated with the policy

vNIC Configuration

Manual vNICs Placement | Auto vNICs Placement

Add

Graphic vNICs Editor

Search | Filters | 4 results | Export

Name	Slot ID	Switch ID	PCI Order	Failover	Pin Group	MAC Pool	vNIC Template	Template Sync ...
eno7	Auto	B	2	Disabled	-	AA02-OCP-MA...	OCP-iSCSI-NV...	OK
eno8	Auto	B	3	Enabled	-	AA02-OCP-MA...	OCP-NFS-vNIC	OK
eno5	Auto	A	0	Enabled	-	AA02-OCP-MA...	OCP-BareMeta...	OK
eno6	Auto	A	1	Disabled	-	AA02-OCP-MA...	OCP-iSCSI-NV...	OK

Step 8. Click **Save** to save the policy.

Procedure 10. Create the OCP Worker Server Profile Template

The OCP Master Server Profile Template can be cloned and modified to create the OCP Worker Server Profile Template.

Step 1. Log into **Cisco Intersight** and select **Infrastructure Service** > **Templates** > **UCS Server Profile Templates.**

Step 2. To the right of the OCP-Master-X210C-M7 template, click **...** and select **Clone**.

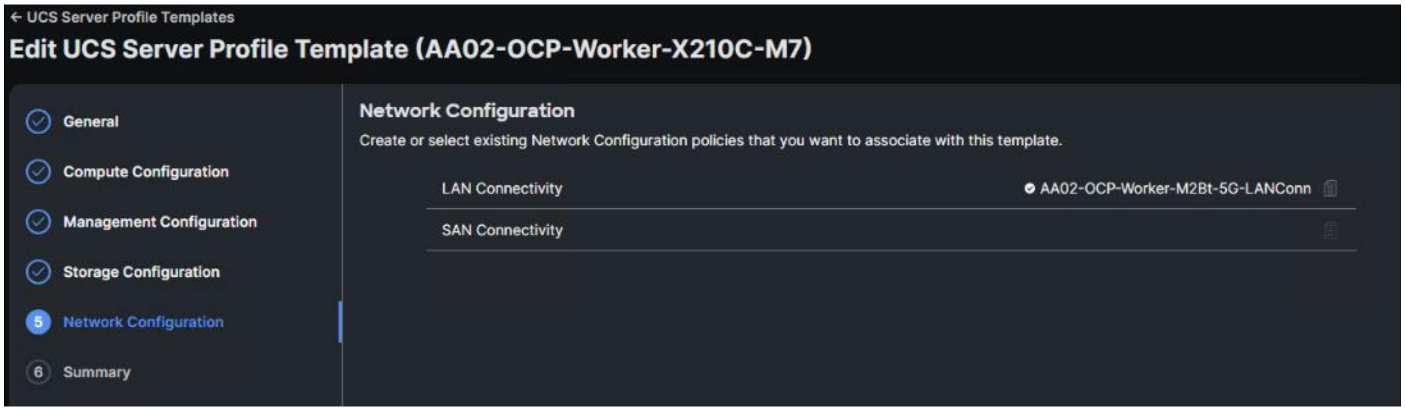
Step 3. Ensure that the correct Destination Organization is selected (for example, AA02-OCP) and click **Next**.

Step 4. Adjust the Clone Name (for example, AA02-Worker-X210C-M7) and Description as needed and click **Next**.

Step 5. From the Templates window, click the **...** to the right of the newly created clone and click **Edit.**

Step 6. Click **Next** until you get to **Storage Configuration.** If the Storage Policy needs to be added or deleted, make that adjustment.

Step 7. Click **Next** to get to Network Configuration. Click the page icon to the right of the LAN Connectivity Policy and select the Worker LAN Connectivity Policy. Click **Select.**


← UCS Server Profile Templates

Edit UCS Server Profile Template (AA02-OCP-Worker-X210C-M7)

Network Configuration

Create or select existing Network Configuration policies that you want to associate with this template.

LAN Connectivity	AA02-OCP-Worker-M2Bt-5G-LANConn
SAN Connectivity	

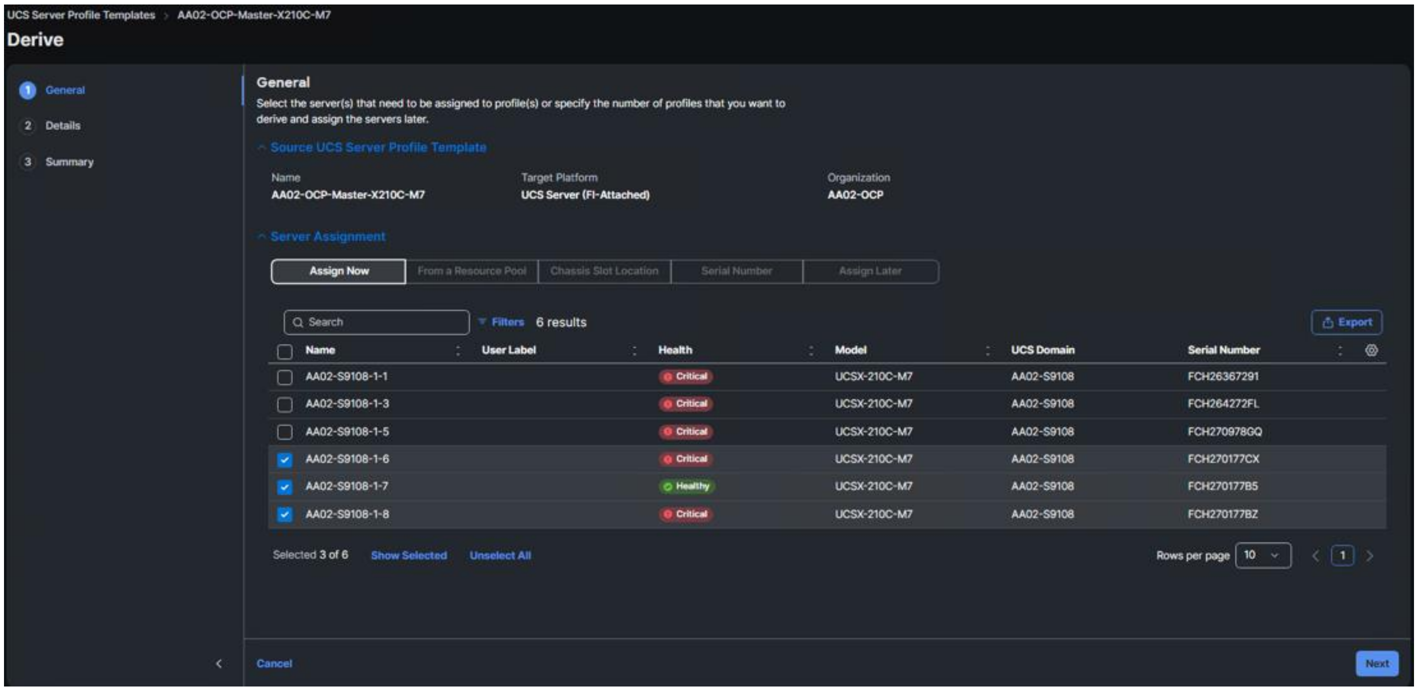
Step 8. Click **Next** and **Close** to save this template.

Complete the Cisco UCS IMM Setup

Procedure 1. Derive Server Profiles

Step 1. From the **Infrastructure Service** > **Templates** page, to the right of the OCP-Master template, click **...** and select **Derive Profiles**.

Step 2. Under the Server Assignment, select **Assign Now** and select the 3 Cisco UCS X210c M7 servers that will be used as OCP Master Nodes.


UCS Server Profile Templates > AA02-OCP-Master-X210C-M7

Derive

General

Select the server(s) that need to be assigned to profile(s) or specify the number of profiles that you want to derive and assign the servers later.

Source UCS Server Profile Template

Name	Target Platform	Organization
AA02-OCP-Master-X210C-M7	UCS Server (FI-Attached)	AA02-OCP

Server Assignment: Assign Now

6 results

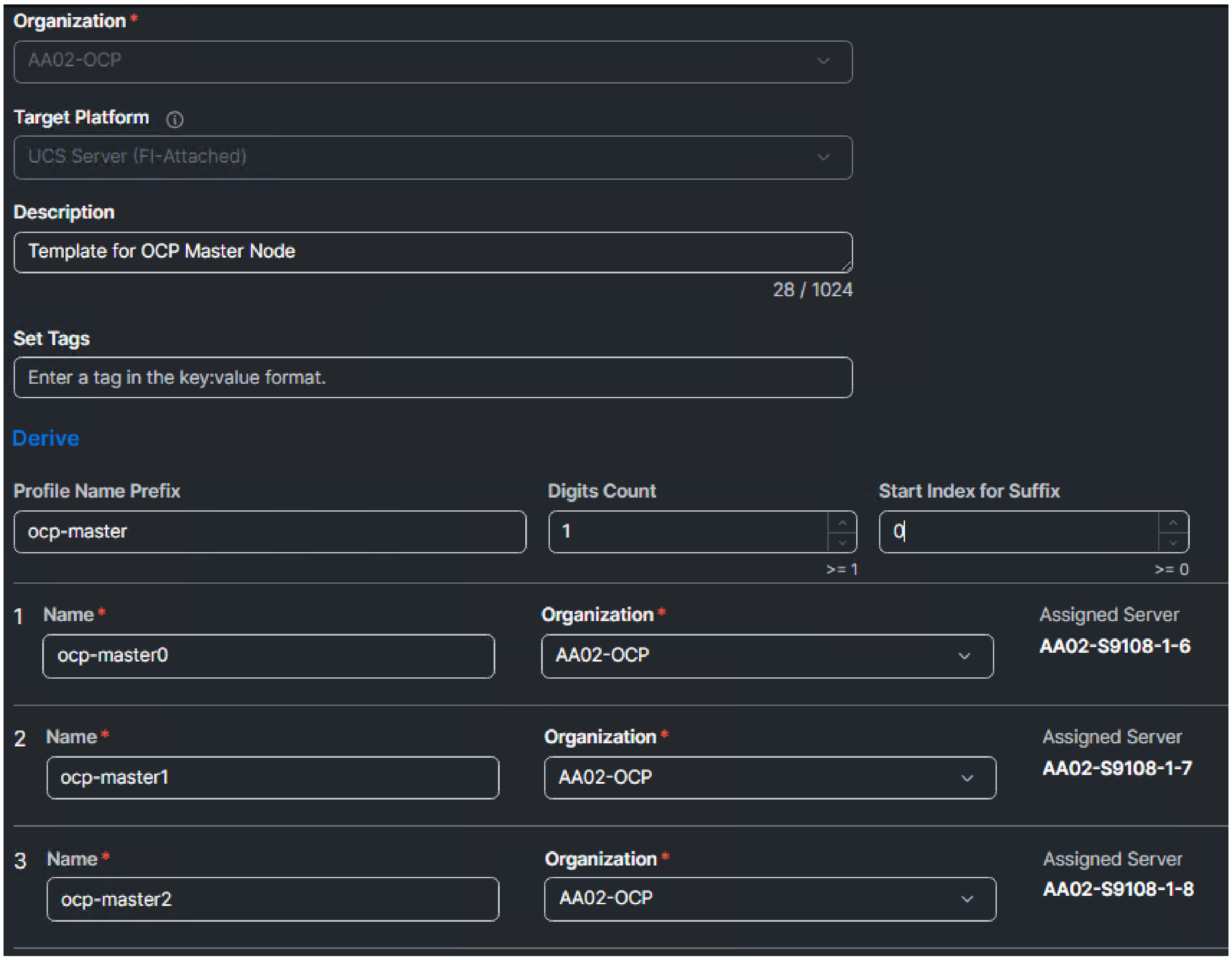
	Name	User Label	Health	Model	UCS Domain	Serial Number
	AA02-S9108-1-1		Critical	UCSX-210C-M7	AA02-S9108	FCH26367291
	AA02-S9108-1-3		Critical	UCSX-210C-M7	AA02-S9108	FCH264272FL
	AA02-S9108-1-5		Critical	UCSX-210C-M7	AA02-S9108	FCH270978GQ
✓	AA02-S9108-1-6		Critical	UCSX-210C-M7	AA02-S9108	FCH270177CX
✓	AA02-S9108-1-7		Healthy	UCSX-210C-M7	AA02-S9108	FCH270177B5
✓	AA02-S9108-1-8		Critical	UCSX-210C-M7	AA02-S9108	FCH270177BZ

Selected 3 of 6

Rows per page 10

Step 3. Click **Next.**

Step 4. For the Profile Name Prefix, enter the first part of the OCP Master Node hostnames (for example, ocp-master. Set Start Index for Suffix to 0 (zero). The 3 server Names should now correspond to the OCP Master Node hostnames.


Organization *

AA02-OCP

Target Platform

UCS Server (FI-Attached)

Description

Template for OCP Master Node

28 / 1024

Set Tags

Enter a tag in the key:value format.

Derive

Profile Name Prefix	Digits Count	Start Index for Suffix
ocp-master	1	0
	>= 1	>= 0

	Name *	Organization *	Assigned Server
1	ocp-master0	AA02-OCP	AA02-S9108-1-6
2	ocp-master1	AA02-OCP	AA02-S9108-1-7
3	ocp-master2	AA02-OCP	AA02-S9108-1-8

Step 5. Click **Next**.

Step 6. Click **Derive** to derive the OCP Master Node Server Profiles.

Step 7. Select **Profiles** on the left and then select the **UCS Server Profiles** tab.

Step 8. Select the 3 OCP Master profiles and then click the **...** at the top or bottom of the list and select **Deploy**.

Step 9. Select **Reboot Immediately to Activate** and click **Deploy**.

Step 10. Repeat this process to create 3 OCP Worker Node Server Profiles.

OpenShift Container Platform Installation and Configuration

This chapter contains the following:

- OpenShift Container Platform – Installation Requirements
- Prerequisites
- Network Requirements
- Deploy NetApp Astra Trident
- NetApp DataOps Toolkit

OCP 4.16 is deployed on the Cisco UCS infrastructure as M.2 booted bare metal servers. The Cisco UCS X210C M7 servers need to be equipped with an M.2 controller (SATA or NVMe) card and either 1 or 2 identical M.2 drives. Three master nodes and three worker nodes are deployed in the validation environment and additional worker nodes can easily be added to increase the scalability of the solution. This document will guide you through the process of using the Assisted Installer to deploy OCP 4.16.

OpenShift Container Platform – Installation Requirements

The Red Hat OCP Assisted Installer provides support for installing OpenShift Container Platform on bare metal nodes. This guide provides a methodology to achieving a successful installation using the Assisted Installer.

Prerequisites

The FlexPod for OCP utilizes the Assisted Installer for OCP installation therefore when provisioning and managing the FlexPod infrastructure, you must provide all the supporting cluster infrastructure and resources, including an installer VM or host, networking, storage, and individual cluster machines.

The following supporting cluster resources are required for the Assisted Installer installation:

- The control plane and compute machines that make up the cluster
- Cluster networking
- Storage for the cluster infrastructure and applications
- The Installer VM or Host

Network Requirements

The following infrastructure services need to be deployed to support the OCP cluster, during the validation of this solution we have provided VMs on your hypervisor of choice to run the required services. Customers can use existing DNS and DHCP services available in the data center:

There are various infrastructure services prerequisites for deploying OCP 4.16. These prerequisites are as follows:

- DNS and DHCP services – these services were configured on Microsoft Windows Server VMs in this validation

- NTP Distribution was done with Nexus switches
- Specific DNS entries for deploying OCP – added to the DNS server
- A Linux VM for initial automated installation and cluster management – a Rocky Linux / RHEL VM with appropriate packages

NTP

Each OpenShift Container Platform node in the cluster must have access to at least two NTP servers.

NICs

NICs configured on the Cisco UCS servers based on the design previously discussed.

DNS

Clients access the OpenShift Container Platform cluster nodes over the bare metal network. Configure a sub-domain or subzone where the canonical name extension is the cluster name.

The following domain and OCP cluster names are used in this deployment guide:

- Base Domain: flexpodb4.cisco.com
- OCP Cluster Name: ocp

The DNS domain name for the OCP cluster should be the cluster name followed by the base domain, for example, ocp.flexpodb4.cisco.com.

Table 11 lists the information for fully qualified domain names used during validation. The API and Nameserver addresses begin with canonical name extensions. The hostnames of the control plane and worker nodes are exemplary, so you can use any host naming convention you prefer.

Table 11. DNS FQDN Names Used

Usage	Hostname	IP Address
API	api.ocp.flexpodb4.cisco.com	10.102.2.228
Ingress LB (apps)	*.apps.ocp.flexpodb4.cisco.com	10.102.2.229
ocp-master0	ocp-master0.ocp.flexpodb4.cisco.com	10.102.2.211
ocp-master1	ocp-master1.ocp.flexpodb4.cisco.com	10.102.2.212
ocp-master2	ocp-master2.ocp.flexpodb4.cisco.com	10.102.2.213
ocp-worker0	ocp-worker0.ocp.flexpodb4.cisco.com	10.102.2.214
ocp-worker1	ocp-worker1.ocpflexpodb4.cisco.com	10.102.2.215

Usage	Hostname	IP Address
ocp-worker2	ocp-worker2.ocp.flexpodb4.cisco.com	10.102.2.216

DHCP

For the bare metal network, a network administrator must reserve several IP addresses, including:

- One IP address for the API endpoint
- One IP address for the wildcard Ingress endpoint
- One IP address for each master node (DHCP server assigns to the node)
- One IP address for each worker node (DHCP server assigns to the node)

Note: Get the MAC addresses of the bare metal Interfaces from the UCS Server Profile for each node to be used in the DHCP configuration to assign reserved IP addresses to the nodes. The KVM IP address also needs to be gathered for the master and worker nodes from the server profiles.

Procedure 1. Gather MAC Addresses of Node Bare Metal Interfaces

Step 1. Log into **Cisco Intersight**.

Step 2. Select **Infrastructure Service** > **Profiles** > **Server Profile** (for example, ocp-worker2).

Step 3. In the center pane, select **Inventory > Network Adapters > Network Adapter** (for example, UCSX-ML-V5D200G).

Step 4. In the center pane, select **Interfaces**.

Step 5. Record the MAC address for NIC Interface eno5.

Step 6. Select the **General** tab and select Identifiers in the center pane.

Step 7. Record the Management IP assigned from the OCP-BareMetal-IP-Pool.

Table 12 lists the IP addresses used for the OCP cluster including bare metal network IPs and UCS KVM Management IPs for IPMI or Redfish access.

Table 12. Host BMC Information

Hostname	IP Address	UCS KVM Mgmt. IP Address	BareMetal MAC Address (eno5)
ocp-master0.ocp.flexpodb4.cisco.com	10.102.2.211	10.102.2.243	00:25:B5:A2:0A:0D
ocp-master1.ocp.flexpodb4.cisco.com	10.102.2.212	10.102.2.244	00:25:B5:A2:0A:09
ocp-master2.ocp.flexpodb4.cisco.com	10.102.2.213	10.102.2.240	00:25:B5:A2:0A:0B
ocp-worker0.ocp.flexpodb4.cisco.com	10.102.2.214	10.102.2.239	00:25:B5:A2:0A:12
ocp-worker1.ocp.flexpodb4.cisco.com	10.102.2.215	10.102.2.242	00:25:B5:A2:0A:13

Hostname	IP Address	UCS KVM Mgmt. IP Address	BareMetal MAC Address (eno5)
ocp-worker2.ocp.flexpodb4.cisco.com	10.102.2.216	10.102.2.241	00:25:B5:A2:0A:14

Step 8. From Table 12, enter the hostnames, IP addresses, and MAC addresses as reservations in your DHCP and DNS server(s) or configure the DHCP server to dynamically update DNS.

Step 9. You will need to pipe VLAN interfaces for all 5 storage VLANs into your DHCP server(s) and assign IPs in the storage networks on those interfaces. Then create a DHCP scope for each storage VLAN and subnet where the IPs assigned by the scope do not overlap with storage LIF IPs.

Step 10. Set up either a VM or spare server as an OCP-Installer machine with the network interface connected to the Bare Metal VLAN and install either Red Hat Enterprise Linux (RHEL) 9.4 or Rocky Linux 9.4 "Server with GUI" and create an administrator user. Once the VM or host is up and running, update it and install and configure XRDP. Also, install Google Chrome onto this machine. Connect to this host with a Windows Remote Desktop client as the admin user.

Procedure 2. Install Red Hat OCP using the Assisted Installer

Step 1. From the Installer desktop, open a terminal session and create an SSH key pair to use to communicate with the OCP hosts:

```
ssh-keygen -t ed25519 -N '' -f ~/.ssh/id_ed25519
```

Step 2. Copy the public SSH key to the user directory:

```
cp ~/.ssh/id_ed25519.pub ~/
```

Step 3. Add the private key to the ssh-agent:

```
ssh add ~/.ssh/id_ed25519
```

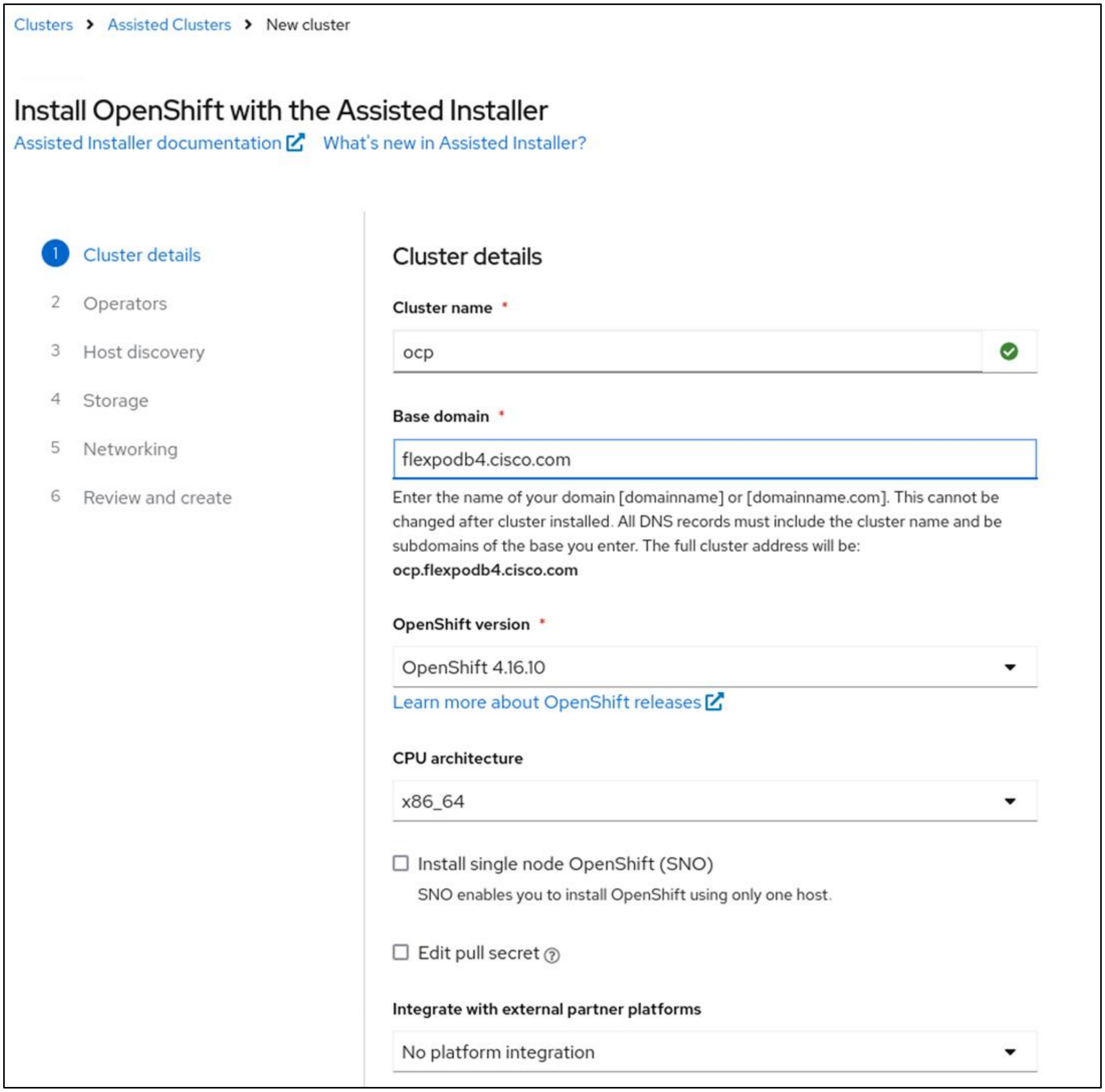
Step 4. Launch Chrome and connect to https://console.redhat.com/openshift/cluster-list. Log into your Red Hat account.

Step 5. Click **Create cluster** to create an OCP cluster.

Step 6. Select **Datacenter** and then select **Bare Metal** (**x86_64**).

Step 7. Select **Interactive** to launch the Assisted Installer.

Step 8. Provide the cluster name and base domain. Scroll down and click **Next**.



Clusters > Assisted Clusters > New cluster

Install OpenShift with the Assisted Installer

Assisted Installer documentation | What's new in Assisted Installer?

Cluster details

Cluster name *: ocp

Base domain *: flexpodb4.cisco.com

Enter the name of your domain [domainname] or [domainname.com]. This cannot be changed after cluster installed. All DNS records must include the cluster name and be subdomains of the base you enter. The full cluster address will be: **ocp.flexpodb4.cisco.com**

OpenShift version *: OpenShift 4.16.10

Learn more about OpenShift releases

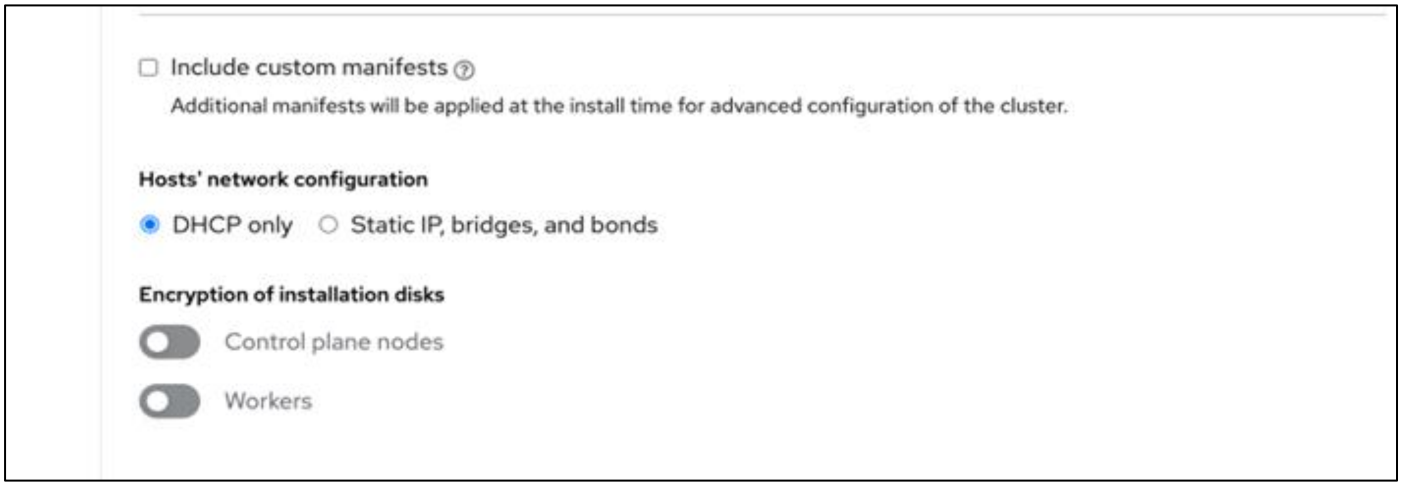
CPU architecture: x86_64

☐ Install single node OpenShift (SNO)

SNO enables you to install OpenShift using only one host.

☐ Edit pull secret

Integrate with external partner platforms: No platform integration



☐ Include custom manifests

Additional manifests will be applied at the install time for advanced configuration of the cluster.

Hosts' network configuration

● DHCP only ○ Static IP, bridges, and bonds

Encryption of installation disks

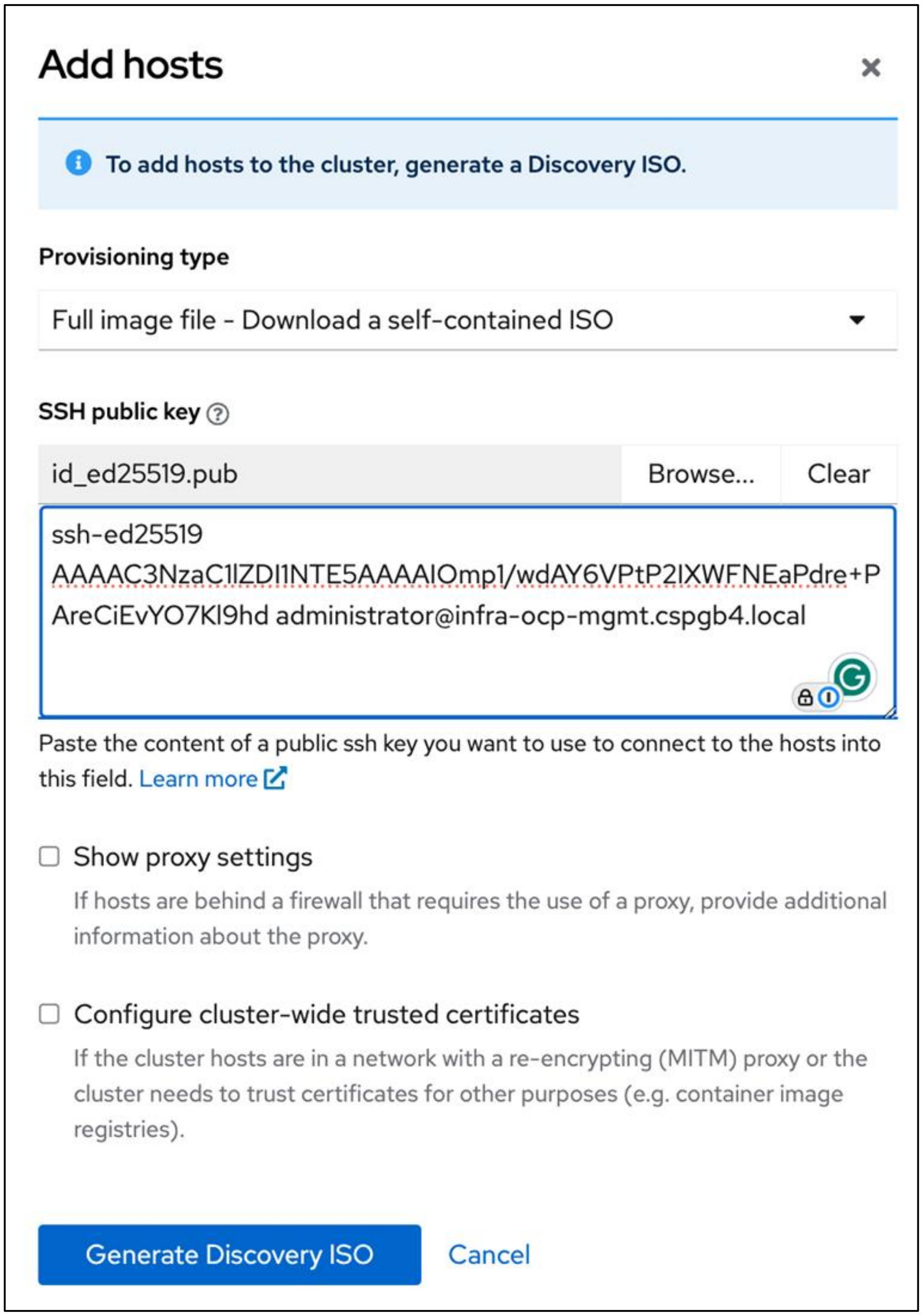
Control plane nodes

Workers

Step 9. It is not necessary to install any Operators at this time, they can be added later. Click **Next.**

Step 10. Click **Add hosts**.

Step 11. Under Provisioning type, from the drop-down list select the Full image file. Under SSH public key, click **Browse** and browse to, select, and open the id_ed25519.pub file**.** The contents of the public key should now appear in the box. Click **Generate Discovery ISO**.



Add hosts

To add hosts to the cluster, generate a Discovery ISO.

Provisioning type

Full image file - Download a self-contained ISO

SSH public key

id_ed25519.pub

```
ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIOmp1/wdAY6VPtP2IXWFNEaPdre+P
AreCiEvYO7KI9hd administrator@infra-ocp-mgmt.cspgb4.local
```

Paste the content of a public ssh key you want to use to connect to the hosts into this field. Learn more

- [ ] Show proxy settings
  If hosts are behind a firewall that requires the use of a proxy, provide additional information about the proxy.
- [ ] Configure cluster-wide trusted certificates
  If the cluster hosts are in a network with a re-encrypting (MITM) proxy or the cluster needs to trust certificates for other purposes (e.g. container image registries).

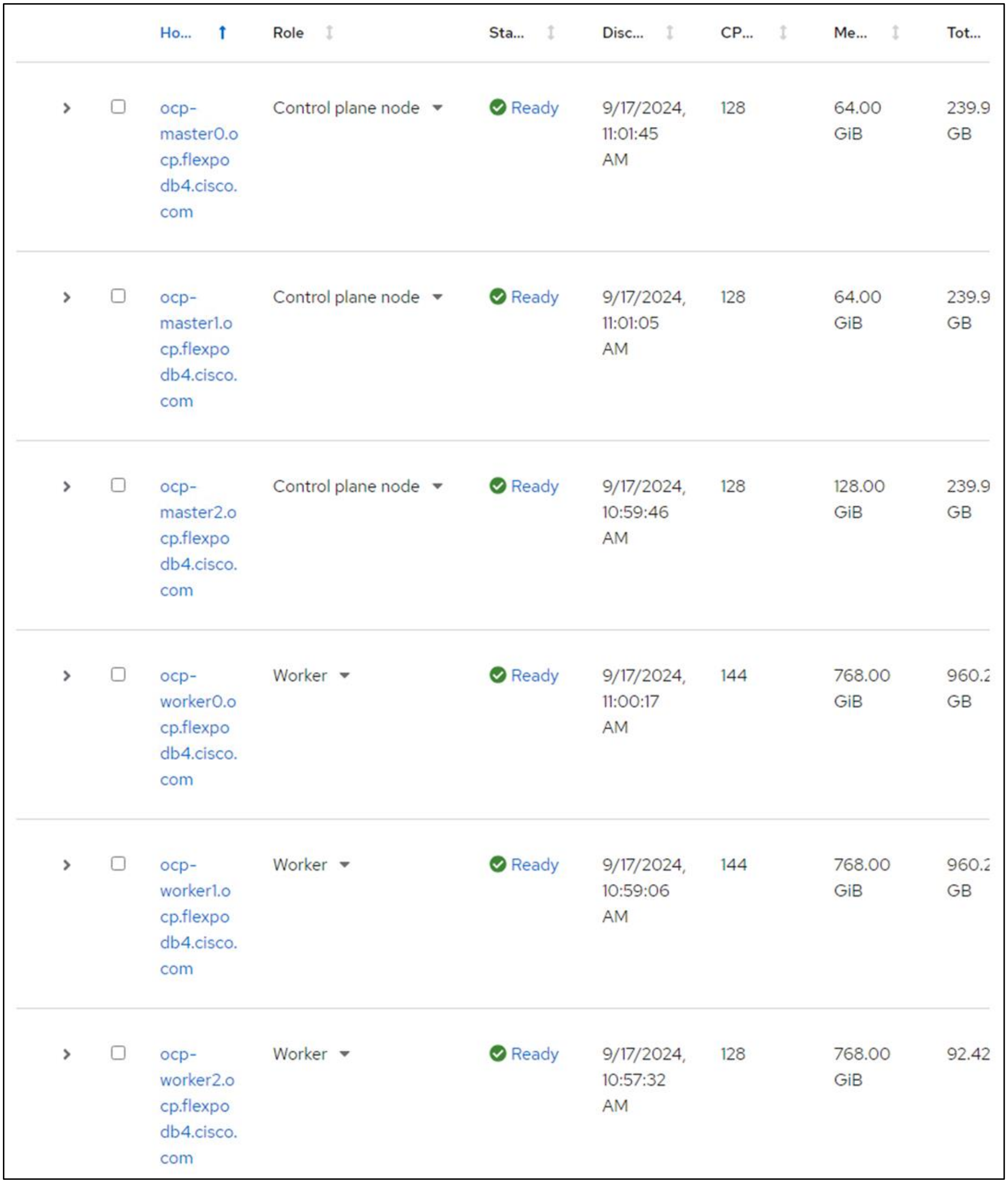
Step 12. Click **Download Discovery ISO** to download the Discover ISO into the Downloads directory. Click **Close** when the download is done.

Step 13. Use Chrome to connect to Cisco Intersight and log into the Intersight account set up earlier.

Step 14. Go to **Infrastructure Service > Operate > Servers** and open a vKVM session on each of the six servers. If the servers are powered on, each server's console should be connected to the Cisco UEFI Interactive Shell.

Step 15. In each of the six vKVMs, click **Virtual Media** and select **vKVM-Mapped DVD**. Map the Discovery ISO from the Downloads folder. Go to **Power > Reset System** and then click **Confirm** to boot the server from the Discovery ISO. Boot all six servers from the Discovery ISO.

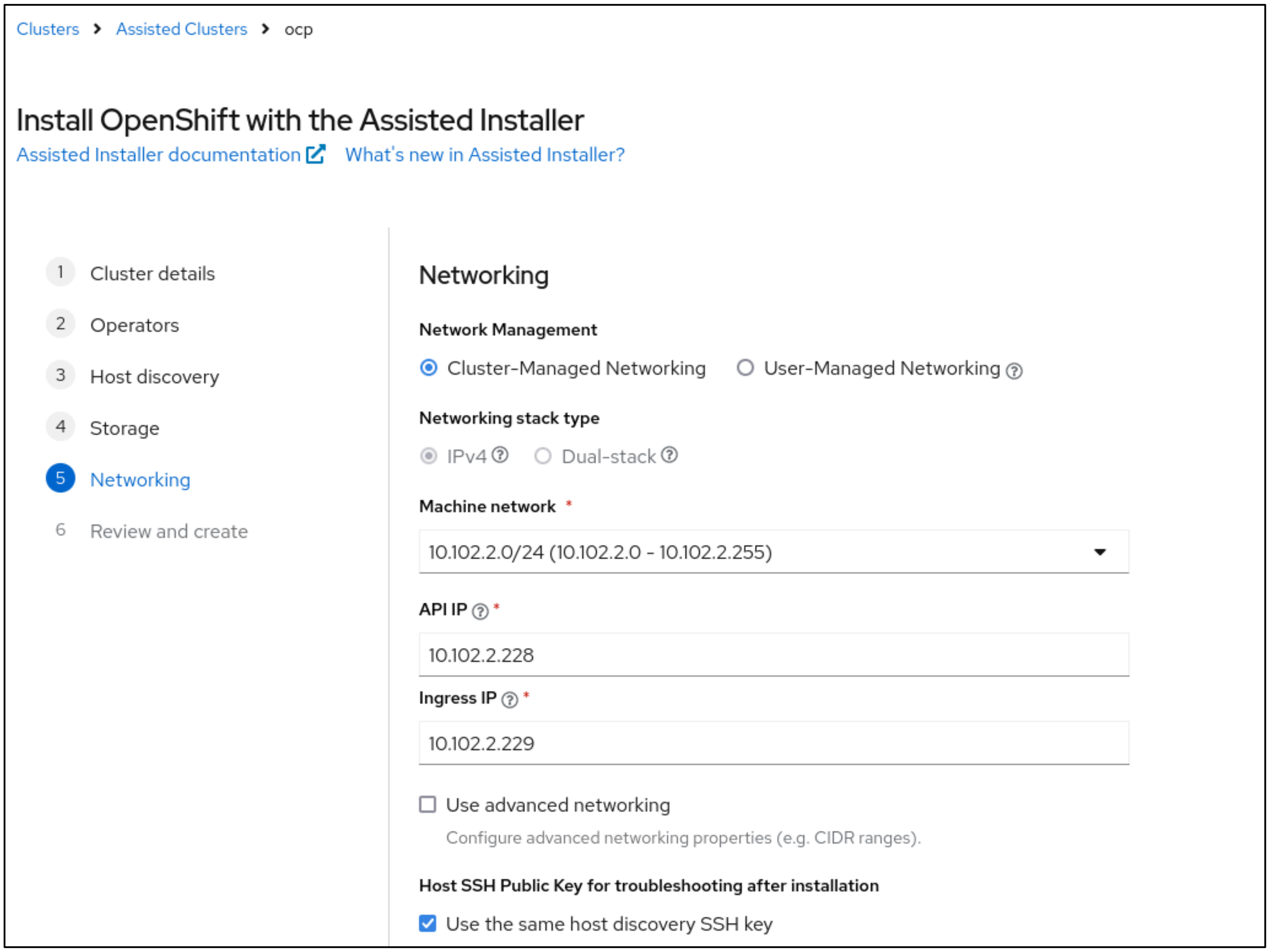
Step 16. Once all six servers have booted "RHEL CoreOS (Live)" from the Discovery ISO, they will appear in the Assisted Installer. From the drop-down lists under Role, assign the appropriate server roles. Scroll down and click **Next**.



		Ho...	Role	Sta...	Disc...	CP...	Me...	Tot...
›	☐	ocp-master0.ocp.flexpodb4.cisco.com	Control plane node	✓ Ready	9/17/2024, 11:01:45 AM	128	64.00 GiB	239.9 GB
›	☐	ocp-master1.ocp.flexpodb4.cisco.com	Control plane node	✓ Ready	9/17/2024, 11:01:05 AM	128	64.00 GiB	239.9 GB
›	☐	ocp-master2.ocp.flexpodb4.cisco.com	Control plane node	✓ Ready	9/17/2024, 10:59:46 AM	128	128.00 GiB	239.9 GB
›	☐	ocp-worker0.ocp.flexpodb4.cisco.com	Worker	✓ Ready	9/17/2024, 11:00:17 AM	144	768.00 GiB	960.2 GB
›	☐	ocp-worker1.ocp.flexpodb4.cisco.com	Worker	✓ Ready	9/17/2024, 10:59:06 AM	144	768.00 GiB	960.2 GB
›	☐	ocp-worker2.ocp.flexpodb4.cisco.com	Worker	✓ Ready	9/17/2024, 10:57:32 AM	128	768.00 GiB	92.42

Step 17. Expand each node and verify CoreOS and OCP is being installed to sda (the M.2 device). Click **Next**.

Step 18. Under **Network Management**, make sure **Cluster-Managed Networking** is selected. Under Machine network, from the drop-down list select the subnet for the BareMetal VLAN. Enter the API IP for the api.cluster.basedomain entry in the DNS servers. For the Ingress IP, enter the IP for the *.apps.cluster.basedomain entry in the DNS servers.


Clusters > Assisted Clusters > ocp
Install OpenShift with the Assisted Installer
Assisted Installer documentation
What's new in Assisted Installer?
1 Cluster details
2 Operators
3 Host discovery
4 Storage
5 Networking
6 Review and create

Networking

Network Management

[✓ Cluster-Managed Networking] User-Managed Networking

Networking stack type

[✓ IPv4] Dual-stack

Machine network *

10.102.2.0/24 (10.102.2.0 - 10.102.2.255)

API IP *

10.102.2.228

Ingress IP *

10.102.2.229

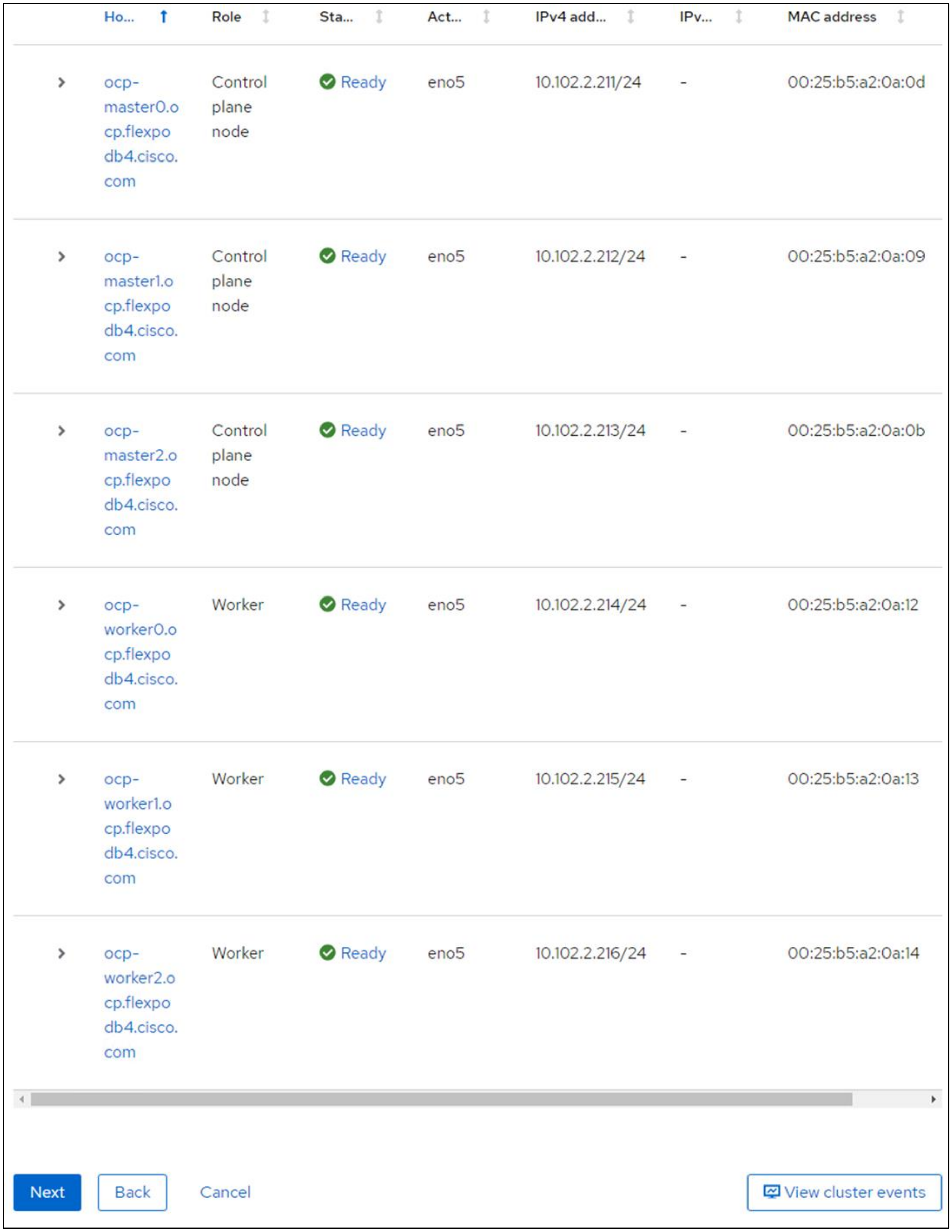
☐ Use advanced networking

Configure advanced networking properties (e.g. CIDR ranges).

Host SSH Public Key for troubleshooting after installation

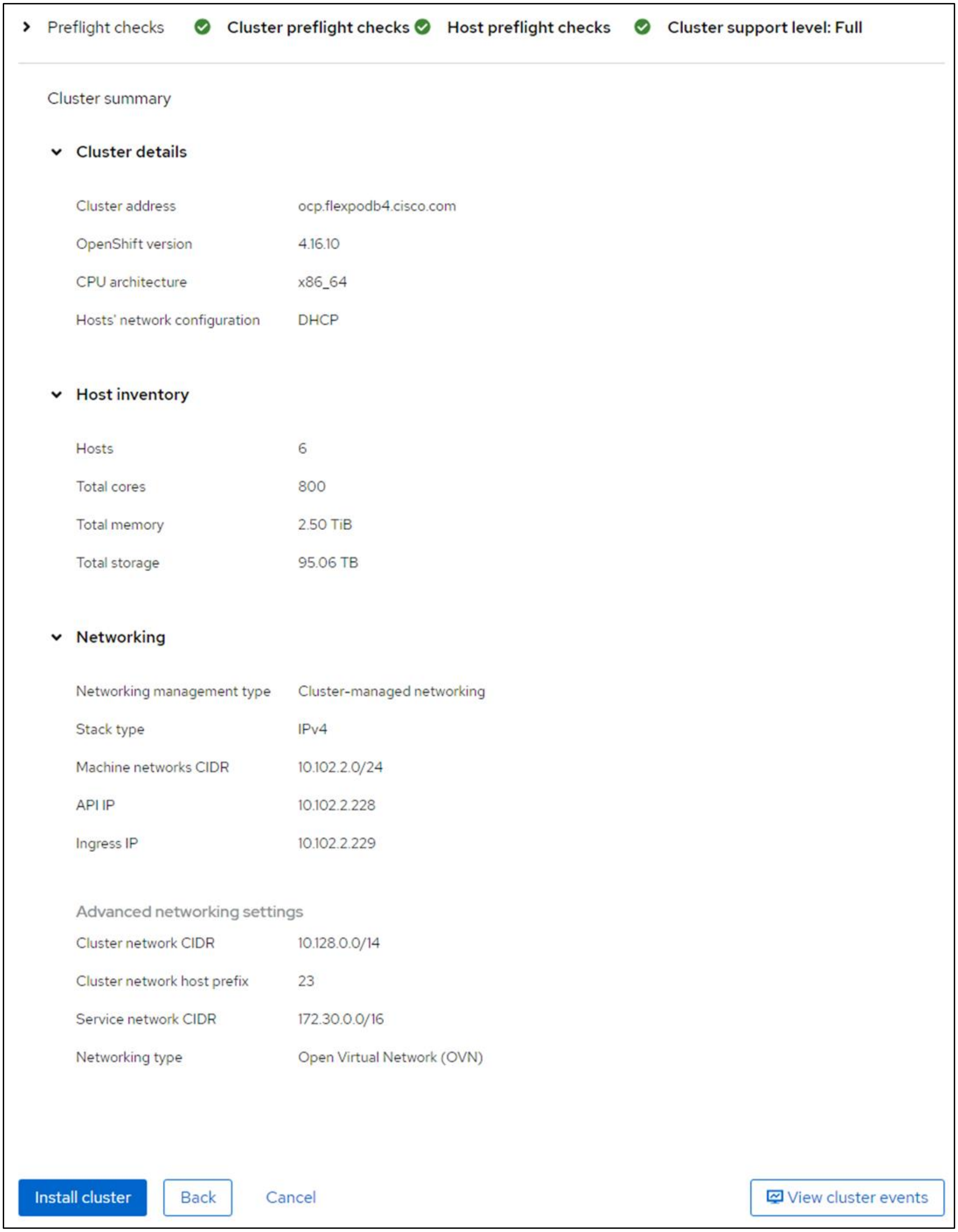
✓ Use the same host discovery SSH key

Step 19. Scroll down. All nodes should all have a status of Ready. Click **Next**.



	Ho...	Role	Sta...	Act...	IPv4 add...	IPv...	MAC address
>	ocp-master0.ocp.flexpodb4.cisco.com	Control plane node	✓ Ready	eno5	10.102.2.211/24	-	00:25:b5:a2:0a:0d
>	ocp-master1.ocp.flexpodb4.cisco.com	Control plane node	✓ Ready	eno5	10.102.2.212/24	-	00:25:b5:a2:0a:09
>	ocp-master2.ocp.flexpodb4.cisco.com	Control plane node	✓ Ready	eno5	10.102.2.213/24	-	00:25:b5:a2:0a:0b
>	ocp-worker0.ocp.flexpodb4.cisco.com	Worker	✓ Ready	eno5	10.102.2.214/24	-	00:25:b5:a2:0a:12
>	ocp-worker1.ocp.flexpodb4.cisco.com	Worker	✓ Ready	eno5	10.102.2.215/24	-	00:25:b5:a2:0a:13
>	ocp-worker2.ocp.flexpodb4.cisco.com	Worker	✓ Ready	eno5	10.102.2.216/24	-	00:25:b5:a2:0a:14

Step 20. Review the information and click **Install cluster** to begin the cluster installation.



Preflight checks ✓ Cluster preflight checks ✓ Host preflight checks ✓ Cluster support level: Full

Cluster summary

Cluster details

Cluster address	ocp.flexpodb4.cisco.com
OpenShift version	4.16.10
CPU architecture	x86_64
Hosts' network configuration	DHCP

Host inventory

Hosts	6
Total cores	800
Total memory	2.50 TiB
Total storage	95.06 TB

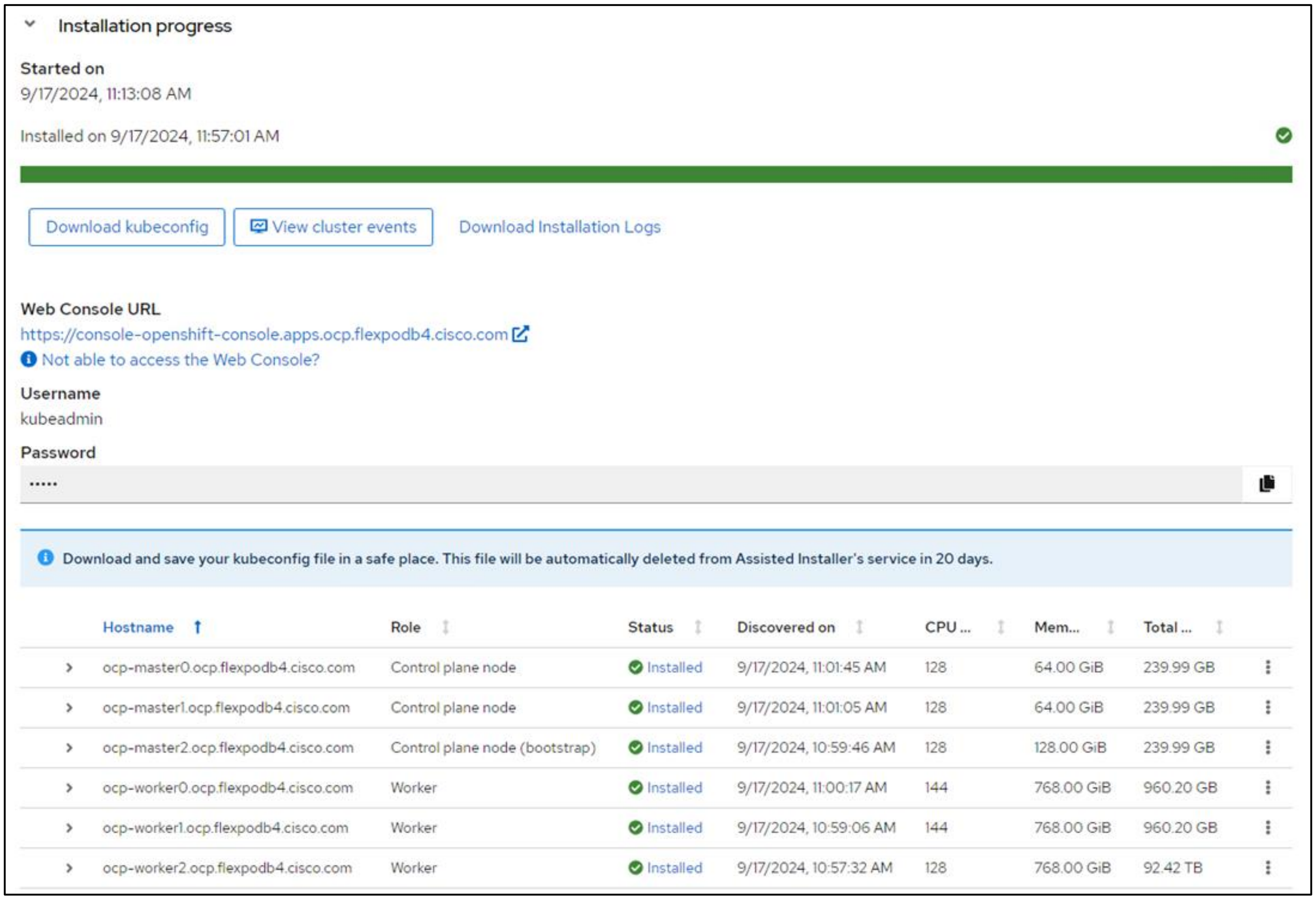
Networking

Networking management type	Cluster-managed networking
Stack type	IPv4
Machine networks CIDR	10.102.2.0/24
API IP	10.102.2.228
Ingress IP	10.102.2.229

Advanced networking settings

Cluster network CIDR	10.128.0.0/14
Cluster network host prefix	23
Service network CIDR	172.30.0.0/16
Networking type	Open Virtual Network (OVN)

Step 21. On the Installation progress page, expand the Host inventory. The installation will take 30-45 minutes. When installation is complete, all nodes displays a Status of Installed.


Installation progress

Started on
9/17/2024, 11:13:08 AM

Installed on 9/17/2024, 11:57:01 AM

Download kubeconfig | View cluster events | Download Installation Logs

Web Console URL
https://console-openshift-console.apps.ocp.flexpodb4.cisco.com
Not able to access the Web Console?

Username
kubeadmin

Password
•••••

Download and save your kubeconfig file in a safe place. This file will be automatically deleted from Assisted Installer's service in 20 days.

Hostname	Role	Status	Discovered on	CPU ...	Mem...	Total ...
ocp-master0.ocp.flexpodb4.cisco.com	Control plane node	Installed	9/17/2024, 11:01:45 AM	128	64.00 GiB	239.99 GB
ocp-master1.ocp.flexpodb4.cisco.com	Control plane node	Installed	9/17/2024, 11:01:05 AM	128	64.00 GiB	239.99 GB
ocp-master2.ocp.flexpodb4.cisco.com	Control plane node (bootstrap)	Installed	9/17/2024, 10:59:46 AM	128	128.00 GiB	239.99 GB
ocp-worker0.ocp.flexpodb4.cisco.com	Worker	Installed	9/17/2024, 11:00:17 AM	144	768.00 GiB	960.20 GB
ocp-worker1.ocp.flexpodb4.cisco.com	Worker	Installed	9/17/2024, 10:59:06 AM	144	768.00 GiB	960.20 GB
ocp-worker2.ocp.flexpodb4.cisco.com	Worker	Installed	9/17/2024, 10:57:32 AM	128	768.00 GiB	92.42 TB

Step 22. Select **Download kubeconfig** to download the kubeconfig file. In a terminal window, setup a cluster directory and save credentials:

```
cd
mkdir <clustername> # for example, ocp
cd <clustername>
mkdir auth
cd auth
mv ~/Downloads/kubeconfig ./
mkdir ~/.kube
cp kubeconfig ~/.kube/config
```

Step 23. In the Assisted Installer, click the icon to copy the kubeadmin password:

```
echo <paste password> > ./kubeadmin-password
```

Step 24. In a new tab in Chrome, connect to https://access.redhat.com/downloads/content/290. Download the OpenShift Linux Client:

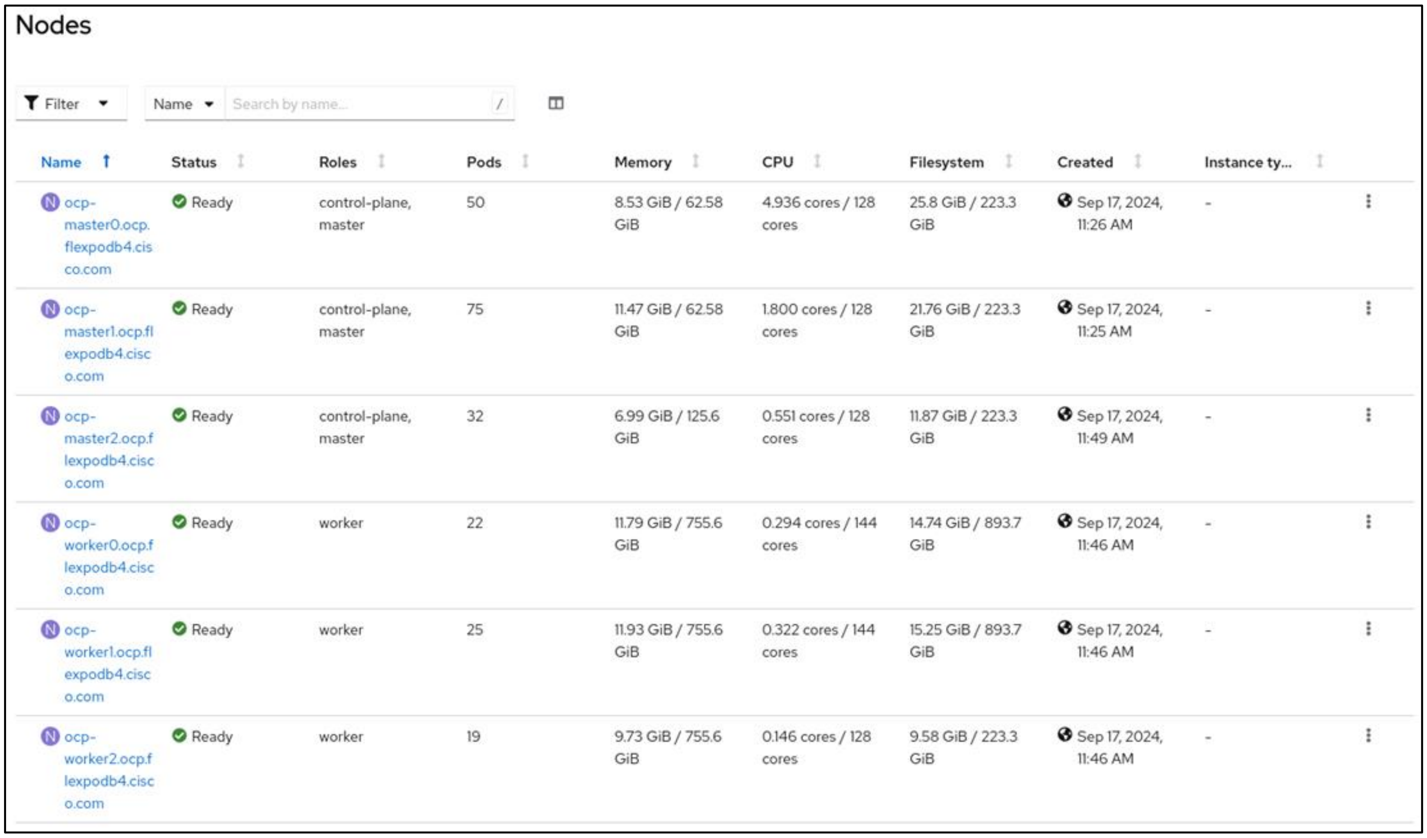
```
cd ..
mkdir client
cd client
ls ~/Downloads
mv ~/Downloads/oc-x.xx.x-linux.tar.gz ./
```

```
tar xvf oc-x.xx.x-linux.tar.gz
ls
sudo mv oc /usr/local/bin/
sudo mv kubectl /usr/local/bin/
oc get nodes
```

Step 25. To enable oc tab completion for bash, run the following:

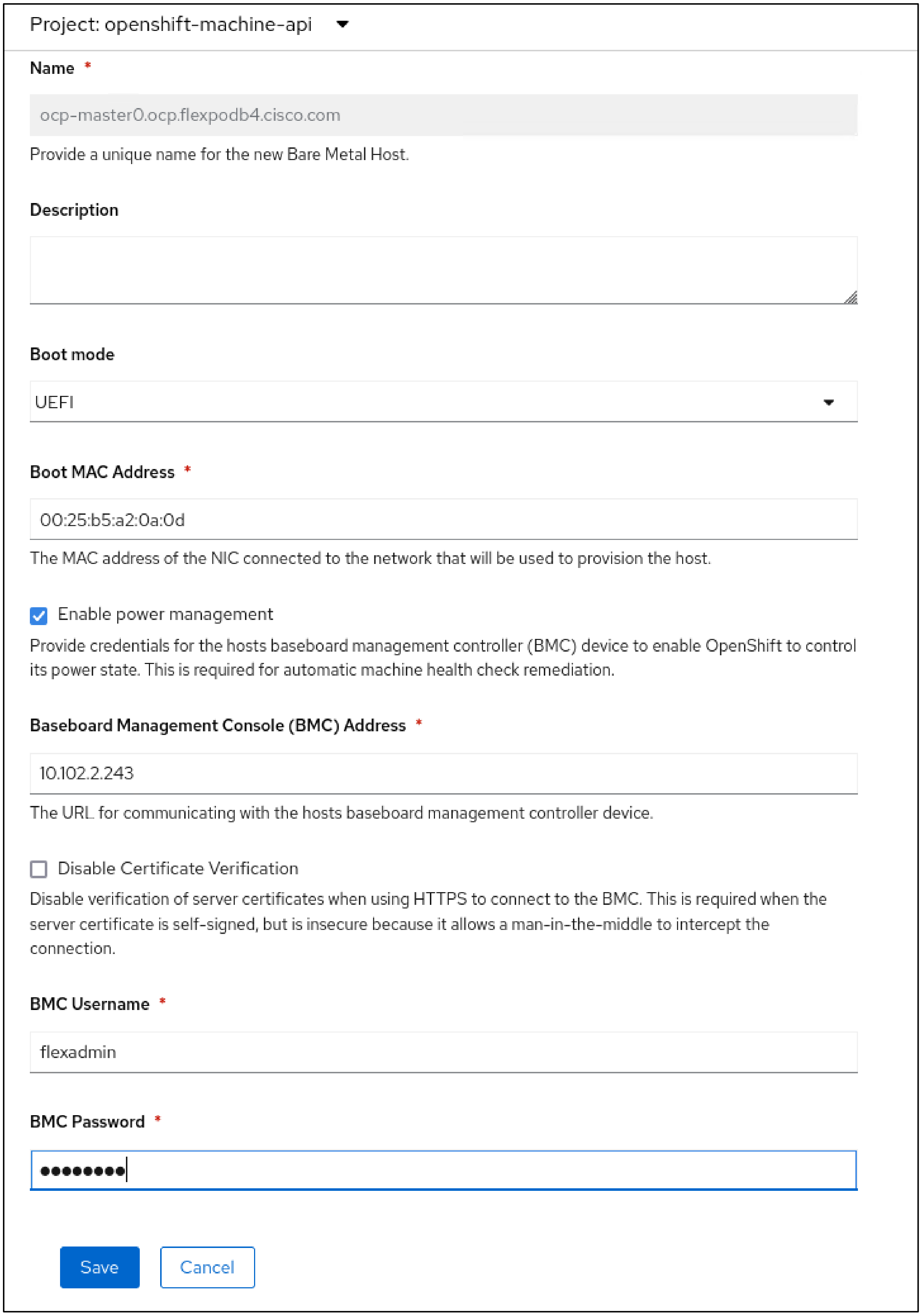
```
oc completion bash > oc_bash_completion
sudo mv oc_bash_completion /etc/bash_completion.d/
```

Step 26. In Chrome, in the Assisted Installer page, click **Launch OpenShift Console** to launch the OCP Console. Use kubeadmin and the kubeadmin password to login. On the left, select **Compute > Nodes** to see the status of the OCP nodes.



Name	Status	Roles	Pods	Memory	CPU	Filesystem	Created	Instance ty...
ocp-master0.ocp.flexpodb4.cisco.com	Ready	control-plane, master	50	8.53 GiB / 62.58 GiB	4.936 cores / 128 cores	25.8 GiB / 223.3 GiB	Sep 17, 2024, 11:26 AM	-
ocp-master1.ocp.flexpodb4.cisco.com	Ready	control-plane, master	75	11.47 GiB / 62.58 GiB	1.800 cores / 128 cores	21.76 GiB / 223.3 GiB	Sep 17, 2024, 11:25 AM	-
ocp-master2.ocp.flexpodb4.cisco.com	Ready	control-plane, master	32	6.99 GiB / 125.6 GiB	0.551 cores / 128 cores	11.87 GiB / 223.3 GiB	Sep 17, 2024, 11:49 AM	-
ocp-worker0.ocp.flexpodb4.cisco.com	Ready	worker	22	11.79 GiB / 755.6 GiB	0.294 cores / 144 cores	14.74 GiB / 893.7 GiB	Sep 17, 2024, 11:46 AM	-
ocp-worker1.ocp.flexpodb4.cisco.com	Ready	worker	25	11.93 GiB / 755.6 GiB	0.322 cores / 144 cores	15.25 GiB / 893.7 GiB	Sep 17, 2024, 11:46 AM	-
ocp-worker2.ocp.flexpodb4.cisco.com	Ready	worker	19	9.73 GiB / 755.6 GiB	0.146 cores / 128 cores	9.58 GiB / 223.3 GiB	Sep 17, 2024, 11:46 AM	-

Step 27. In the **Red Hat OpenShift console**, go to **Compute > Bare Metal Hosts.** For each Bare Metal Host, click the ellipses to the right of the host and select **Edit Bare Metal Host**. Select **Enable power management**. Using Table12, fill in the BMC Address. Also, make sure the Boot MAC Address matches the MAC address in Table12. For the BMC Username and BMC Password, use what was entered into the Cisco Intersight IPMI over LAN policy. Click **Save** to save the changes. Repeat this step for all Bare Metal Hosts.


Project: openshift-machine-api

Name *

ocp-master0.ocp.flexpodb4.cisco.com

Provide a unique name for the new Bare Metal Host.

Description

Boot mode

UEFI

Boot MAC Address *

00:25:b5:a2:0a:0d

The MAC address of the NIC connected to the network that will be used to provision the host.

[✓] Enable power management

Provide credentials for the hosts baseboard management controller (BMC) device to enable OpenShift to control its power state. This is required for automatic machine health check remediation.

Baseboard Management Console (BMC) Address *

10.102.2.243

The URL for communicating with the hosts baseboard management controller device.

[ ] Disable Certificate Verification

Disable verification of server certificates when using HTTPS to connect to the BMC. This is required when the server certificate is self-signed, but is insecure because it allows a man-in-the-middle to intercept the connection.

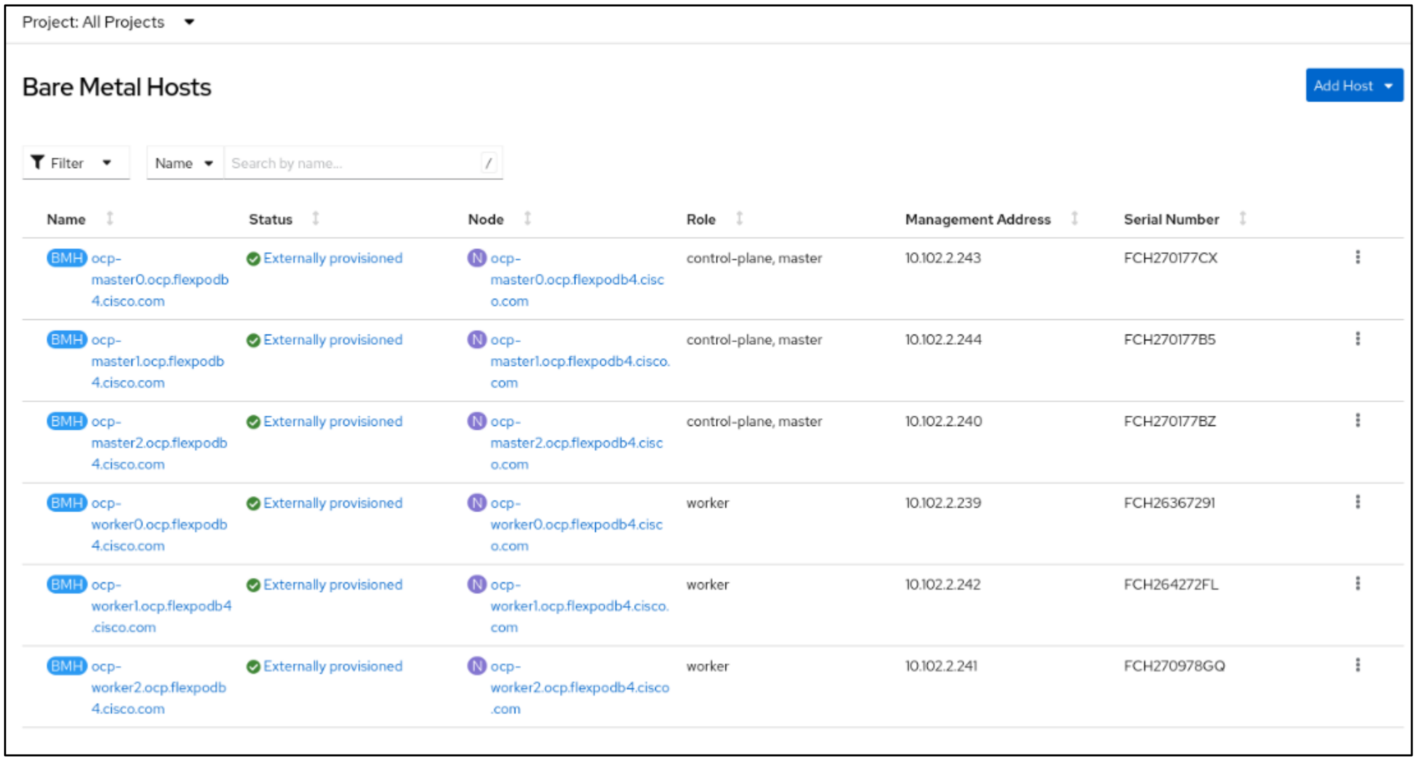
BMC Username *

flexadmin

BMC Password *

●●●●●●●●

Step 28. Go to **Compute > Bare Metal Hosts.** Once all hosts have been configured. The Status should show “Externally provisioned,” and the Management Address should be populated. You can now manage power on the OCP hosts from the OCP console.



Project: All Projects

Bare Metal Hosts

Name	Status	Node	Role	Management Address	Serial Number
BMH ocp-master0.ocp.flexpodb4.cisco.com	Externally provisioned	N ocp-master0.ocp.flexpodb4.cisco.com	control-plane, master	10.102.2.243	FCH270177CX
BMH ocp-master1.ocp.flexpodb4.cisco.com	Externally provisioned	N ocp-master1.ocp.flexpodb4.cisco.com	control-plane, master	10.102.2.244	FCH270177B5
BMH ocp-master2.ocp.flexpodb4.cisco.com	Externally provisioned	N ocp-master2.ocp.flexpodb4.cisco.com	control-plane, master	10.102.2.240	FCH270177BZ
BMH ocp-worker0.ocp.flexpodb4.cisco.com	Externally provisioned	N ocp-worker0.ocp.flexpodb4.cisco.com	worker	10.102.2.239	FCH26367291
BMH ocp-worker1.ocp.flexpodb4.cisco.com	Externally provisioned	N ocp-worker1.ocp.flexpodb4.cisco.com	worker	10.102.2.242	FCH264272FL
BMH ocp-worker2.ocp.flexpodb4.cisco.com	Externally provisioned	N ocp-worker2.ocp.flexpodb4.cisco.com	worker	10.102.2.241	FCH270978GQ

Step 29. To configure storage networking on the worker nodes, NTP on the worker and master nodes, and NVMe-TCP on the worker nodes, run the following:

```
cd
cd <cluster-name> # For example, ocp
mkdir machine-configs
cd machine-configs
curl https://mirror.openshift.com/pub/openshift-v4/clients/butane/latest/butane --output butane
chmod +x butane
```

Step 30. Build the following files in the machine-configs directory with variations for your network:

Note: It is critical that the NetworkManager configuration files have the mode “0600” to set the permissions of the file correctly.

```
cat 99-worker-eno6.bu
variant: openshift
version: 4.16.0
metadata:
  name: 99-worker-eno6
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/NetworkManager/system-connections/eno6.nmconnection
      mode: 0600
      overwrite: true
      contents:
        inline: |
          [connection]
          id=eno6
          type=ethernet
          interface-name=eno6

          [ethernet]

          [ipv4]
          method=auto

          [ipv6]
          addr-gen-mode=default
          method=disabled

          [proxy]

          [user]

cat 99-worker-eno6-3032.bu
variant: openshift
version: 4.16.0
metadata:
  name: 99-worker-eno6-3032
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/NetworkManager/system-connections/eno6-3032.nmconnection
      mode: 0600
      overwrite: true
      contents:
        inline: |
          [connection]
          id=eno6-3032
          type=vlan
          interface-name=eno6-3032

          [ethernet]

          [vlan]
          flags=1
          id=3032
          parent=eno6

          [ipv4]
          method=auto

          [ipv6]
          addr-gen-mode=default
          method=disabled

          [proxy]

cat 99-worker-eno7.bu
variant: openshift
version: 4.16.0
metadata:
  name: 99-worker-eno7
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/NetworkManager/system-connections/eno7.nmconnection
      mode: 0600
      overwrite: true
      contents:
        inline: |
          [connection]
          id=eno7
          type=ethernet
          interface-name=eno7
          [ethernet]

          [ipv4]
          method=auto

          [ipv6]
          addr-gen-mode=default
          method=disabled

          [proxy]

          [user]

cat 99-worker-eno7-3042.bu
variant: openshift
version: 4.16.0
metadata:
  name: 99-worker-eno7-3042
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/NetworkManager/system-connections/eno7-3042.nmconnection
      mode: 0600
      overwrite: true
      contents:
        inline: |
          [connection]
          id=eno7-3042
          type=vlan
          interface-name=eno7-3042

          [ethernet]

          [vlan]
          flags=1
          id=3042
          parent=eno7

          [ipv4]
          method=auto

          [ipv6]
          addr-gen-mode=default
          method=disabled

          [proxy]

cat 99-worker-eno8.bu
variant: openshift
version: 4.16.0
metadata:
  name: 99-worker-eno8
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/NetworkManager/system-connections/eno8.nmconnection
      mode: 0600
      overwrite: true
      contents:
        inline: |
          [connection]
          id=eno8
          type=ethernet
          interface-name=eno8

          [ethernet]
          [ipv4]
          method=auto

          [ipv6]
          addr-gen-mode=default
          method=disabled

          [proxy]

          [user]

cat 99-master-chrony-conf-override.bu
variant: openshift
version: 4.16.0
metadata:
  name: 99-master-chrony-conf-override
  labels:
    machineconfiguration.openshift.io/role: master
storage:
  files:
    - path: /etc/chrony.conf
      mode: 0644
      overwrite: true
      contents:
        inline: |
          driftfile /var/lib/chrony/drift
          makestep 1.0 3
          rtcsync
          logdir /var/log/chrony
          server 10.102.2.3 iburst
          server 10.102.2.4 iburst

cat 99-worker-chrony-conf-override.bu
variant: openshift
version: 4.16.0
metadata:
  name: 99-worker-chrony-conf-override
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/chrony.conf
      mode: 0644
      overwrite: true
      contents:
        inline: |
          driftfile /var/lib/chrony/drift
          makestep 1.0 3
          rtcsync
          logdir /var/log/chrony
          server 10.102.2.3 iburst
          server 10.102.2.4 iburst

cat 99-worker-nvme-discovery.bu
variant: openshift
version: 4.16.0
metadata:
  name: 99-worker-nvme-discovery
  labels:
    machineconfiguration.openshift.io/role: worker
openshift:
  kernel_arguments:
    - loglevel=7
storage:
  files:
    - path: /etc/nvme/discovery.conf
      mode: 0644
      overwrite: true
      contents:
        inline: |
          --transport=tcp --traddr=192.168.32.51 --trsvcid=8009
          --transport=tcp --traddr=192.168.42.52 --trsvcid=8009
```

Step 31. Create .yaml files from the butane files with butane, then load the configurations into OCP:

```
./butane 99-worker-eno6.bu -o ./99-worker-eno6.yaml
./butane 99-worker-eno6-3032.bu -o ./99-worker-eno6-3032.yaml
./butane 99-worker-eno7.bu -o ./99-worker-eno7.yaml
./butane 99-worker-eno7-3042.bu -o ./99-worker-eno7-3042.yaml
./butane 99-worker-eno8.bu -o ./99-worker-eno8.yaml./butane 99-master-chrony-conf-override.bu
-o ./99-master-chrony-conf-override.yaml
./butane 99-worker-chrony-conf-override.bu -o ./99-worker-chrony-conf-override.yaml
./butane 99-worker-nvme-discovery.bu -o ./99-worker-nvme-discovery.yaml

oc create -f 99-worker-eno6.yaml
oc create -f 99-worker-eno6-3032.yaml
oc create -f 99-worker-eno7.yaml

oc create -f 99-worker-eno7-3042.yaml
oc create -f 99-worker-eno8.yaml
oc create -f 99-master-chrony-conf-override.yaml
oc create -f 99-worker-chrony-conf-override.yaml
oc create -f 99-worker-nvme-discovery.yaml
```

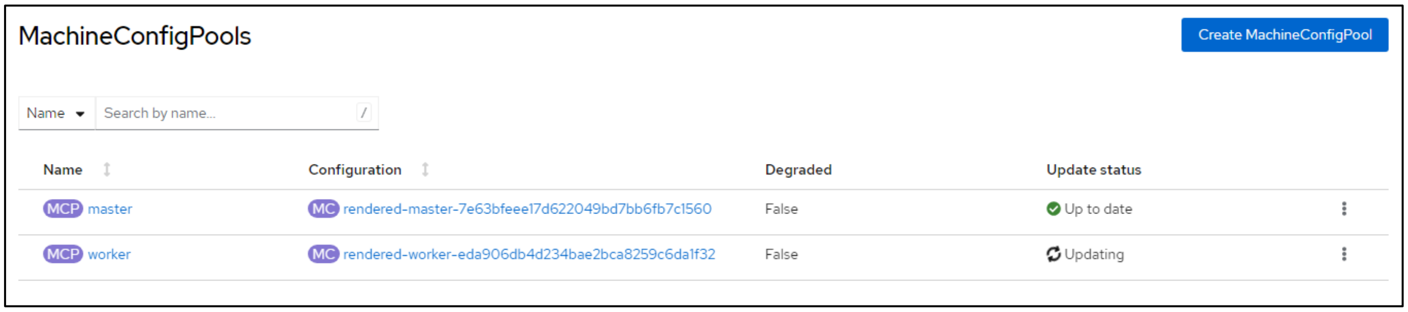
Step 32. To enable iSCSI and multipathing on the workers, create the 99-worker-ontap-iscsi.yaml and upload as a machine config:

```
cat 99-worker-ontap-iscsi.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: 99-worker-ontap-iscsi
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source:
data:text/plain;charset=utf-8;base64,IyBkZXZpY2UtbWFwcGVyLW11bHRpcGF0aCBjb25maWd1cmF0aW9uIGZpbGUKCiMgRm9yIGEg
Y29tcGxldGUgbGlzdCBvZiB0aGUgZGVmYXVsdCBjb25maWd1cmF0aW9uIHZhbHVlcywgcnVuIGVpdGhlcjoKIyAjIG11bHRpcGF0aCAtdAojI
G9yCiMgIyBtdWx0aXBhdGhkIHNob3cgY29uZmlnCgojIEZvciBhIGxpc3Qgb2YgY29uZmlndXJhdGlvbiBvcHRpb25zIHdpdGggZGVzY3JpcH
Rpb25zLCBzZWUgdGhlCiMgbXVsdGlwYXRoLmNvbmYgbWFuIHBhZ2UuCgpkZWZhdWx0cyB7Cgl1c2VyX2ZyaWVuZGx5X25hbWVzIHllcwoJZml
uZF9tdWx0aXBhdGhzIG5vCn0KCmJsYWNrbGlzdCB7Cn0K
          verification: {}
        filesystem: root
        mode: 600
        overwrite: true
        path: /etc/multipath.conf
    systemd:
      units:
        - name: iscsid.service
          enabled: true
          state: started
        - name: multipathd.service
          enabled: true
          state: started
  osImageURL: ""

oc create -f 99-worker-ontap-iscsi.yaml
```

Note: The Base 64 encoded source above is the following file (/etc/multipath.conf) encoded. It is necessary to set “find_multipaths” to no.

```
cat multipath.conf
# device-mapper-multipath configuration file

# For a complete list of the default configuration values, run either:
# # multipath -t
# or
# # multipathd show config

# For a list of configuration options with descriptions, see the
# multipath.conf man page.

defaults {
        user_friendly_names yes
        find_multipaths no
}

blacklist {
}
```

Step 33. Over the next 20-30 minutes each of the nodes will go through the "Not Ready" state and reboot. You can monitor this by going to **Compute** > **MachineConfigPools** in the OCP Console. After each Worker moves back to the "Ready" state, ssh into the Worker and type "ifconfig -a." Verify that the two VLAN interfaces are set up correctly, are receiving IPs, and have an MTU of 9000.



Name	Configuration	Degraded	Update status
MCP master	MC rendered-master-7e63bfeee17d622049bd7bb6fb7c1560	False	Up to date
MCP worker	MC rendered-worker-eda906db4d234bae2bca8259c6da1f32	False	Updating

Procedure 3. Install the NVIDIA GPU Operator

If you have GPUs installed in your Cisco UCS servers, you need to install the Node Feature Discovery (NFD) Operator to detect NVIDIA GPUs and the NVIDIA GPU Operator to make these GPUs available to containers and virtual machines.

Step 1. In the **OpenShift Container Platform web console**, click **Operators > OperatorHub**.

Step 2. Type Node Feature in the Filter box and then click on the Node Feature Discovery Operator with Red Hat in the upper right corner. Click **Install**.

Step 3. Do not change any settings and click **Install**.

Step 4. When the Install operator is ready for use, click **View Operator**.

Step 5. In the bar to the right of Details, click **NodeFeatureDiscovery**.

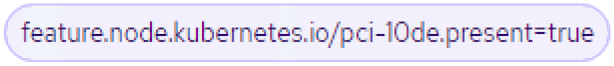
Step 6. Click **Create NodeFeatureDiscovery**.

Step 7. Click **Create**.

Step 8. When the nfd-instance has a status of Available, Upgradeable, select **Compute > Nodes**.

Step 9. Select a node that has one or more GPUs and then select **Details**.

The following label should be present on the host:



feature.node.kubernetes.io/pci-10de.present=true

Note: This label should appear on all nodes with GPUs.

Step 10. Go to **Operators** > **OperatorHub**.

Step 11. Type NVIDIA in the Filter box and then click the NVIDIA GPU Operator. Click **Install**.

Step 12. Do not change any settings and click **Install**.

Step 13. When the Install operator is ready for use, click **View Operator**.

Step 14. In the bar to the right of Details, click **ClusterPolicy**.

Step 15. Click **Create ClusterPolicy**.

Step 16. Do not change any settings and scroll down and click **Create**. This will install the latest GPU driver.

Step 17. Wait for the gpu-cluster-policy Status to become Ready.

Step 18. Connect to a terminal window on the OCP Installer machine. Type the following commands. The output shown is for two servers that are equipped with GPUs:

```
oc project nvidia-gpu-operator
Now using project "nvidia-gpu-operator" on server "https://api.ocp.flexpodb4.cisco.com:6443".

oc get pods
NAME                                                    READY   STATUS      RESTARTS         AGE
gpu-feature-discovery-jmlbr                             1/1     Running     0                6m45s
gpu-feature-discovery-l2l6n                             1/1     Running     0                6m41s
gpu-operator-6656d9fbf-wkkfm                            1/1     Running     0                11m
nvidia-container-toolkit-daemonset-gb8d9                1/1     Running     0                6m45s
nvidia-container-toolkit-daemonset-t4xdf                1/1     Running     0                6m41s
nvidia-cuda-validator-lc8zr                             0/1     Completed   0                4m33s
nvidia-cuda-validator-zxvnx                             0/1     Completed   0                4m39s
nvidia-dcgm-exporter-k6tnp                              1/1     Running     2 (4m7s ago)     6m41s
nvidia-dcgm-exporter-vb66w                              1/1     Running     2 (4m20s ago)    6m45s
nvidia-dcgm-hfgz2                                       1/1     Running     0                6m45s
nvidia-dcgm-qwm46                                       1/1     Running     0                6m41s
nvidia-device-plugin-daemonset-nr6m7                    1/1     Running     0                6m41s
nvidia-device-plugin-daemonset-rpvwr                    1/1     Running     0                6m45s
nvidia-driver-daemonset-416.94.202407231922-0-88zcr     2/2     Running     0                7m42s
nvidia-driver-daemonset-416.94.202407231922-0-bvph6     2/2     Running     0                7m42s
nvidia-node-status-exporter-bz79d                       1/1     Running     0                7m41s
nvidia-node-status-exporter-jgjbd                       1/1     Running     0                7m41s
nvidia-operator-validator-8fxqr                         1/1     Running     0                6m41s
nvidia-operator-validator-tbqtc                         1/1     Running     0                6m45s
```

Step 19. Connect to one of the nvidia-driver-daemonset containers and view the GPU status.

```
oc exec -it nvidia-driver-daemonset-416.94.202407231922-0-88zcr -- bash

[root@nvidia-driver-daemonset-416 drivers]# nvidia-smi
Tue Aug 13 21:19:36 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.90.07              Driver Version: 550.90.07      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA L40S                    On  |   00000000:38:00.0 Off |                    0 |
| N/A   26C    P8             33W /  350W |       1MiB /  46068MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA L40S                    On  |   00000000:D8:00.0 Off |                    0 |
| N/A   25C    P8             34W /  350W |       1MiB /  46068MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

Procedure 4. Enable the GPU Monitoring Dashboard (Optional)

Go to: https://docs.nvidia.com/datacenter/cloud-native/openshift/latest/enable-gpu-monitoring-dashboard.html and enable the GPU Monitoring Dashboard to monitor GPUs in the OpenShift Web-Console.

Deploy NetApp Astra Trident

NetApp Astra Trident is an open-source, fully supported storage orchestrator for containers and Kubernetes distributions. It was designed to help meet the containerized applications' persistence demands using industry-standard interfaces, such as the Container Storage Interface (CSI). With Astra Trident, microservices and containerized applications can take advantage of enterprise-class storage services provided by the NetApp portfolio of storage systems. More information about Trident can be found here: NetApp Trident Documentation. NetApp Astra Trident can be installed via different methods. In this solution we will discuss using helm to install Trident version 24.06.1.

Note: In this solution, we validated NetApp Astra Trident with the ontap-nas driver and ontap-nas-flexgroup driver using the NFS protocol. We also validated the ontap-san driver for iSCSI and NVMe-TCP.

Procedure 1. Install Helm version

Step 1. Download the Helm binary and add it to your path:

```
sudo curl -L https://mirror.openshift.com/pub/openshift-v4/clients/helm/latest/helm-linux-amd64 -o
/usr/local/bin/helm
```

Step 2. Make the binary file executable:

```
sudo chmod +x /usr/local/bin/helm
```

Step 3. Check the installed version:

```
helm version
```

Procedure 2. Install Trident using Helm

Step 1. From the OCP directory, download Trident software from GitHub and untar the .gz file to obtain the trident-installer folder:

```
wget https://github.com/NetApp/trident/releases/download/v24.06.1/trident-installer-24.06.1.tar.gz
tar -xvf trident-installer-24.06.1.tar.gz
cd trident-installer/helm
```

Step 2. Create Trident namespace:

```
oc create namespace trident
```

Step 3. Install Trident using helm:

```
helm install ocp-trident trident-operator-100.2406.1.tgz -n trident
```

Step 4. Check the Trident OCP pods after installation:

```
oc get pods -n trident
NAME                                 READY   STATUS    RESTARTS   AGE
trident-controller-b786b7cdb-dw4zp   6/6     Running   0          4m18s
trident-node-linux-46t42             2/2     Running   0          4m18s
trident-node-linux-5pqss             2/2     Running   0          4m18s
trident-node-linux-9w96c             2/2     Running   0          4m18s
trident-node-linux-pb775             2/2     Running   0          4m18s
trident-node-linux-tdj4r             2/2     Running   0          4m18s
trident-node-linux-zsnnx             2/2     Running   0          4m18s
trident-operator-6c4dd7fd6-vbpbs     1/1     Running   0          4m36s
```

Note: If the Astra Trident deployment fails and does not bring up the pods to Running state, use the `tridentctl logs -l all -n trident` command for debugging.

Note: Before configuring the backends that Trident needs to use for user apps, go to: https://docs.netapp.com/us-en/trident/trident-reference/objects.html#kubernetes-customresourcedefinition-objects to understand the storage environment parameters and its usage in Trident.

Procedure 3. Configure the Storage Backends in Trident

Step 1. Configure the connections to the SVM on the NetApp storage array created for the OCP installation. For more options regarding storage backend configuration, refer to https://docs.netapp.com/us-en/trident/trident-use/backends.html.

Step 2. Create a backends directory and create the following backend definition files in that directory:

```
cat backend_NFS.yaml
---
version: 1
storageDriverName: ontap-nas
backendName: ocp-nfs-backend
managementLIF: 10.102.2.50
dataLIF: 192.168.52.51
svm: OCP-Trident-SVM
username: vsadmin
password: <password>
useREST: true
defaults:
  spaceReserve: none
  exportPolicy: default
  snapshotPolicy: default
  snapshotReserve: '10'

cat backend_NFS_flexgroup.yaml
---
version: 1
storageDriverName: ontap-nas-flexgroup
backendName: ocp-nfs-flexgroup
managementLIF: 10.102.2.50
dataLIF: 192.168.52.51
svm: OCP-Trident-SVM
username: vsadmin
password: <password>
useREST: true
defaults:
  spaceReserve: none
  exportPolicy: default
  snapshotPolicy: default
  snapshotReserve: '10'

cat backend_iSCSI.yaml
```

```
---
version: 1
storageDriverName: ontap-san
backendName: ocp-iscsi-backend
managementLIF: 10.102.2.50
svm: OCP-Trident-SVM
sanType: iscsi
useREST: true
username: vsadmin
password: <password>
defaults:
  spaceReserve: none
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'

cat backend_NVMe.yaml
---
version: 1
backendName: ocp-nvme-backend
storageDriverName: ontap-san
managementLIF: 10.102.2.50
svm: OCP-Trident-SVM
username: vsadmin
password: <password>
sanType: nvme
useREST: true
defaults:
  spaceReserve: none
  snapshotPolicy: default
  snapshotReserve: '10'
```

Step 3. Activate the storage backends for all storage protocols in your FlexPod:

```
tridentctl -n trident create backend -f backend_NFS.yaml
tridentctl -n trident create backend -f backend_NFS_flexgroup.yaml
tridentctl -n trident create backend -f backend_iSCSI.yaml
tridentctl -n trident create backend -f backend_NVMe.yaml
tridentctl -n trident get backend
+-------------------+---------------------+--------------------------------------+--------+------------+-----
|       NAME        |   STORAGE DRIVER    |                 UUID                 | STATE  | USER-STATE | VOLU
+-------------------+---------------------+--------------------------------------+--------+------------+-----
| ocp-nfs-backend   | ontap-nas           | 6bcb2421-a148-40bb-b7a4-9231e58efc2a | online | normal     |
| ocp-nfs-flexgroup | ontap-nas-flexgroup | 68428a01-c5e6-4676-8cb5-e5521fc04bc7 | online | normal     |
| ocp-iscsi-backend | ontap-san           | bbf1664d-1615-42d3-a5ed-1b8aed995a42 | online | normal     |
| ocp-nvme-backend  | ontap-san           | 2b6861a2-6980-449a-b718-97002079e7f3 | online | normal     |
+-------------------+---------------------+--------------------------------------+--------+------------+-----
```

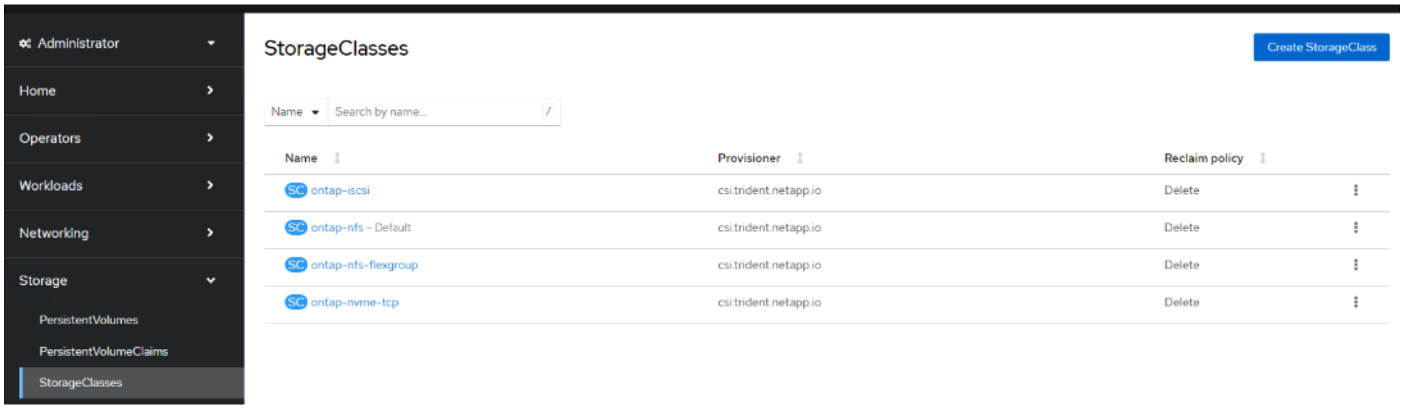
Step 4. Create the following Storage Class files:

```
cat storage-class-ontap-nfs.yaml
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nfs
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  provisioningType: "thin"
  snapshots: "true"
allowVolumeExpansion: true

cat storage-class-ontap-nfs-flexgroup.yaml
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
```

```
metadata:
  name: ontap-nfs-flexgroup
  annotations:
    storageclass.kubernetes.io/is-default-class: "false"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas-flexgroup"
  provisioningType: "thin"
  snapshots: "true"
allowVolumeExpansion: true

cat storage-class-ontap-iscsi.yaml
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-iscsi
parameters:
  backendType: "ontap-san"
  sanType: "iscsi"
  provisioningType: "thin"
  snapshots: "true"
allowVolumeExpansion: true
provisioner: csi.trident.netapp.io

cat storage-class-ontap-nvme.yaml
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nvme-tcp
parameters:
  backendType: "ontap-san"
  sanType: "nvme"
  provisioningType: "thin"
  snapshots: "true"
allowVolumeExpansion: true
provisioner: csi.trident.netapp.io
```

Step 5. Create the storage classes:

```
oc create -f storage-class-ontap-nfs.yaml
oc create -f storage-class-ontap-nfs-flexgroup.yaml
oc create -f storage-class-ontap-iscsi.yaml
oc create -f storage-class-ontap-nvme.yaml
```



StorageClasses

Name	Provisioner	Reclaim policy
SC ontap-iscsi	csi.trident.netapp.io	Delete
SC ontap-nfs – Default	csi.trident.netapp.io	Delete
SC ontap-nfs-flexgroup	csi.trident.netapp.io	Delete
SC ontap-nvme-tcp	csi.trident.netapp.io	Delete

Step 6. Create a VolumeSnapshotClass file:

```
cat ontap-volumesnapshot-class.yaml
---
apiVersion: snapshot.storage.k8s.io/v1
```

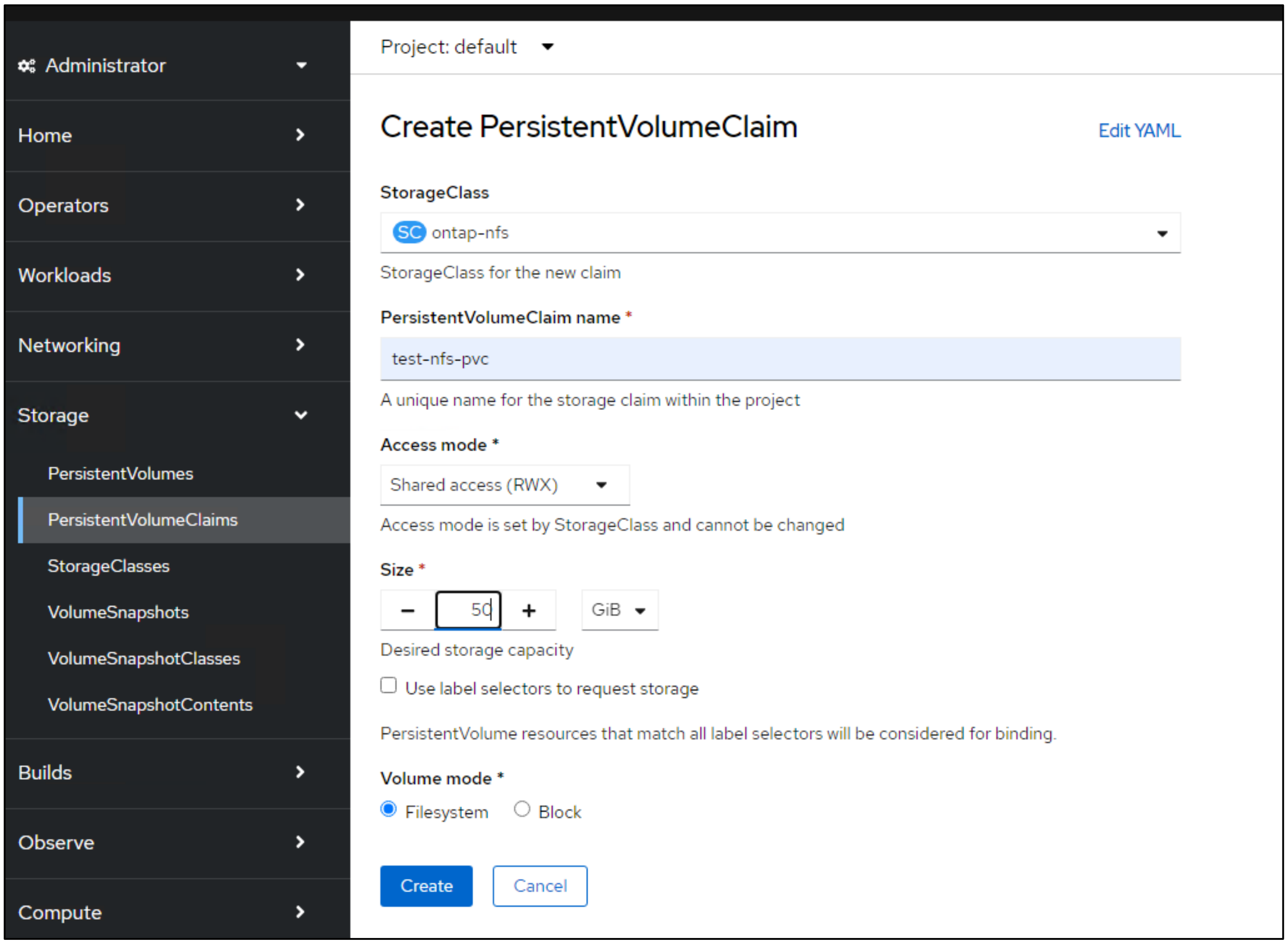
```
kind: VolumeSnapshotClass
metadata:
  name: ontap-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

Step 7. Create the VolumeSnapshotClass:

```
oc create -f ontap-volumesnapshot-class.yaml
```

Step 8. Create a test PersistentVolumeClaim (PVC). In the OCP console, click **Storage > PersistentVolumeClaims**. Select an appropriate project or create a new project and select it. On the right, click **Create PersistentVolumeClaim**.

Step 9. Select a StorageClass and give the PVC a name. Select an Access mode (RWO or RWX for NFS classes, and RWO for iSCSI or NVMe-TCP classes). Set a size and select a Volume mode (normally Filesystem). Click **Create** to create the storage class.



Project: default

Create PersistentVolumeClaim

StorageClass: SC ontap-nfs
StorageClass for the new claim

PersistentVolumeClaim name *: test-nfs-pvc
A unique name for the storage claim within the project

Access mode *: Shared access (RWX)
Access mode is set by StorageClass and cannot be changed

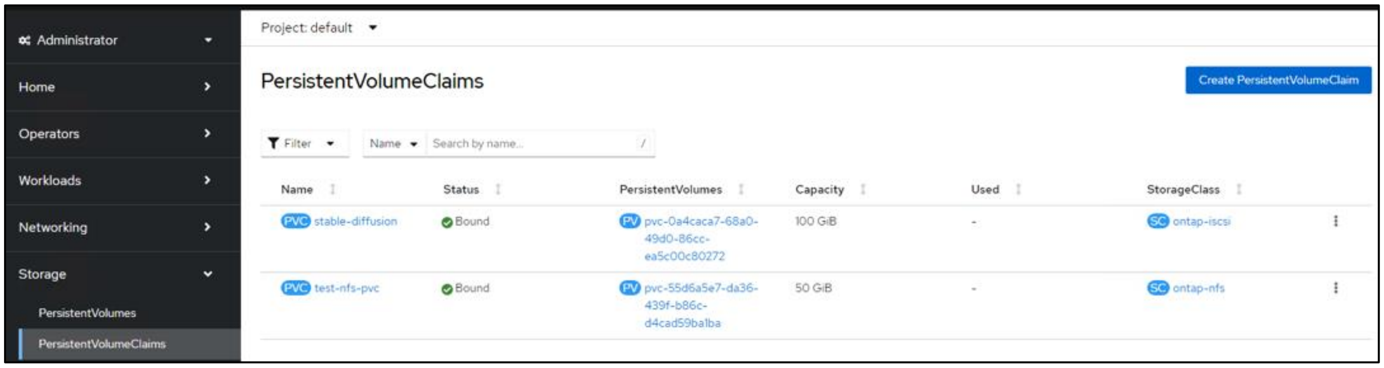
Size *: 50 GiB
Desired storage capacity

☐ Use label selectors to request storage

PersistentVolume resources that match all label selectors will be considered for binding.

Volume mode *: [✓ Filesystem] Block

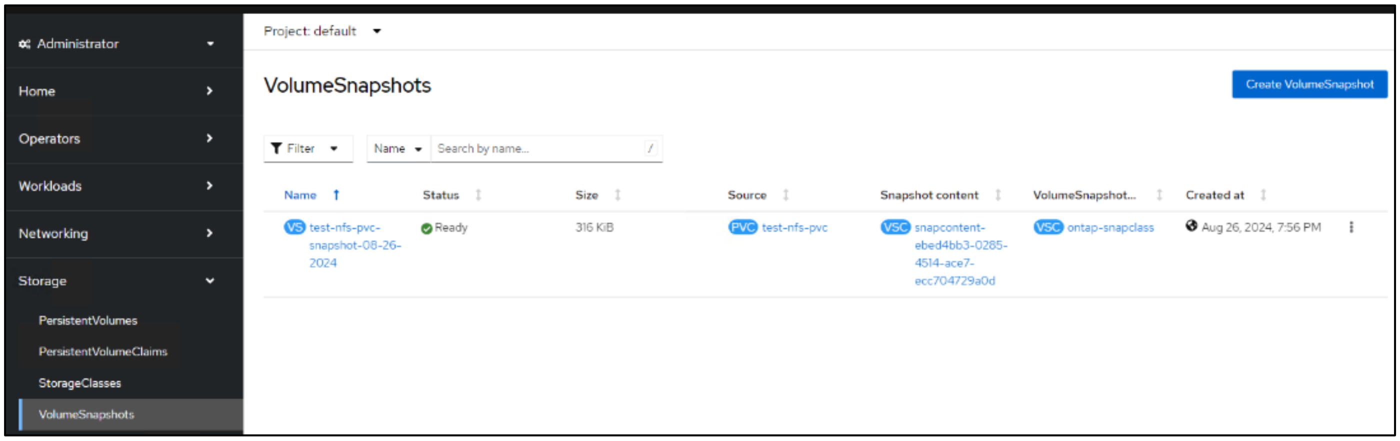
Step 10. Wait for the PVC to have a status of Bound. The PVC can now be attached to a container.



PersistentVolumeClaims

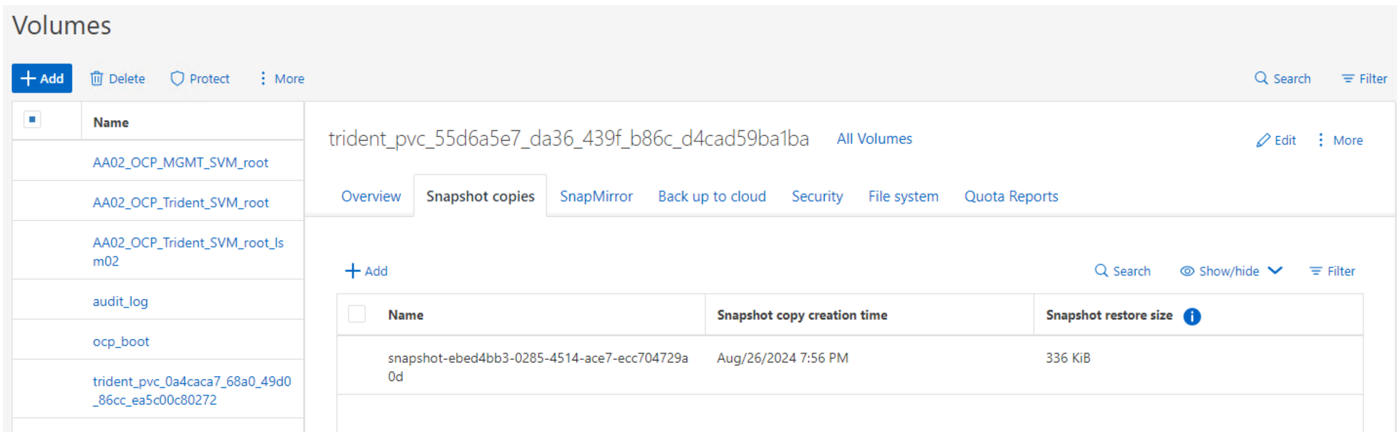
Name	Status	PersistentVolumes	Capacity	Used	StorageClass
PVC stable-diffusion	Bound	PV pvc-0a4caca7-68a0-49d0-86cc-ea5c00c80272	100 GiB	-	SC ontap-iscsi
PVC test-nfs-pvc	Bound	PV pvc-55d6a5e7-da36-439f-b86c-d4cad59ba1ba	50 GiB	-	SC ontap-nfs

Step 11. Create a NetApp volume snapshot of the PVC by clicking the three dots to the right of the PVC and selecting **Create snapshot**. Adjust the snapshot name and click **Create**. The snapshot will appear under VolumeSnapshots and can also be seen in NetApp System Manager under the corresponding PV.



VolumeSnapshots

Name	Status	Size	Source	Snapshot content	VolumeSnapshot...	Created at
VS test-nfs-pvc-snapshot-08-26-2024	Ready	316 KiB	PVC test-nfs-pvc	VSC snapcontent-ebed4bb3-0285-4514-ace7-ecc704729a0d	VSC ontap-snapclass	Aug 26, 2024, 7:56 PM



Volumes

Name
AA02_OCP_MGMT_SVM_root
AA02_OCP_Trident_SVM_root
AA02_OCP_Trident_SVM_root_lsm02
audit_log
ocp_boot
trident_pvc_0a4caca7_68a0_49d0_86cc_ea5c00c80272

trident_pvc_55d6a5e7_da36_439f_b86c_d4cad59ba1ba

Name	Snapshot copy creation time	Snapshot restore size
snapshot-ebed4bb3-0285-4514-ace7-ecc704729a0d	Aug/26/2024 7:56 PM	336 KiB

NetApp DataOps Toolkit

The version 2.5.0 toolkit is currently compatible with Kubernetes versions 1.20 and above, and OpenShift versions 4.7 and above.

The toolkit is currently compatible with Trident versions 20.07 and above. Additionally, the toolkit is compatible with the following Trident backend types used in this validation:

- ontap-nas
- ontap-nas-flexgroup

More operations and capabilities about NetApp DataOps Toolkit are available and documented here: https://github.com/NetApp/netapp-dataops-toolkit

Prerequisites

The NetApp DataOps Toolkit for Kubernetes requires that Python 3.8 or above be installed on the local host. Additionally, the toolkit requires that pip for Python3 be installed on the local host. For more details regarding pip, including installation instructions, refer to the pip documentation.

Procedure 1. NetApp DataOps Toolkit Installation

Step 1. To install the NetApp DataOps Toolkit for Kubernetes on the OCP Installer VM, run the following command:

```
python3 -m pip install netapp-dataops-k8s
```

Note: NetApp DataOps Toolkit is used to create jupyterlab, clone jupyterlab, create a snapshot for a JupyterLab workspace, and so on.

Note: You can use NetApp DataOps Toolkit to create Jupyter notebooks in this solution. For more information, go to: Create a new JupyterLab workspace.

RAG Pipeline Installation and Configuration

This chapter contains the following:

- Canonical RAG
- Multi-turn RAG

Canonical RAG and Multi-turn RAG were deployed as example RAG pipelines. They are part of the Enterprise RAG 24.06 Release from NVIDIA. These pipelines are designed to enhance retrieval-augmented generation (RAG) applications, providing robust solutions for enterprise needs. Early access information and further details can be found on the NVIDIA NeMo Microservices page - https://developer.nvidia.com/nemo-microservices

Canonical RAG

This example showcases RAG pipeline. It uses NVIDIA NIM for Large Language Models microservice to host TRT optimized LLM and NVIDIA NeMo Retriever Embedding Microservice. It uses Milvus as vectorstore to store embeddings and generate response for query.

LLM Model	Embedding	Framework	Document Type	Vector Database	Model deployment platform
meta/llama3-8b-instruct	snowflake-arctic-embed-l	llama-index	PDF/Text	milvus	On Prem

Prerequisites

- You have the NGC CLI available on your client machine. You can download the CLI from https://ngc.nvidia.com/setup/installers/cli.
- Setup your NGC API Key: https://org.ngc.nvidia.com/setup
- You have a default storage class available in the OpenShift cluster for PVC provisioning.

Procedure 1. Deploy NVIDIA NIM for LLM

NVIDIA NIM for LLMs Helm Chart simplifies NIM deployment on Kubernetes. It aims to support deployment with a variety of possible cluster, GPU, and storage configurations.

Step 1. Set the NGC_API_KEY environment variable to your NGC API key, as shown in the following example:

```
export NGC_API_KEY="key from ngc"
```

Step 2. Create a namespace:

```
oc create namespace nim
```

Step 3. Change the namespace to the newly created one:

```
oc project nim
```

Step 4. To download the NIM container image, you must set an image pull secret, which is ngc-secret in the following example. To download model engines or weights from NGC, the chart requires a generic secret that has an NGC API key as a value stored in a key named NGC_API_KEY. The following example creates these two values:

```
oc create secret docker-registry ngc-secret --docker-server=nvcr.io --docker-username='$oauthtoken'
--docker-password=$NGC_API_KEY


oc create secret generic ngc-api --from-literal=NGC_API_KEY=$NGC_API_KEY
```

Step 5. Use the following command to download the helm chart. This downloads the chart as a file to your local machine:

```
helm fetch https://helm.ngc.nvidia.com/nim/charts/nim-llm-1.1.2.tgz --username='$oauthtoken'
--password=$NGC_API_KEY
```

Step 6. Extract the content:

```
tar xvfz nim-llm-1.1.2.tgz
```

Step 7. Go to the "nim-llm" directory and customize the values.yaml file.

The following helm options are the most important options to configure to deploy a NIM using Kubernetes. For more details see: https://docs.nvidia.com/nim/large-language-models/latest/deploy-helm.html

- **image.repository** – The container/NIM to deploy
- **image.tag** – The version of that container/NIM

```
image:
  repository: nvcr.io/nim/meta/llama3-8b-instruct
  pullPolicy: IfNotPresent
  # Tag overrides the image tag whose default is the chart appVersion.
  tag: "1.0.3"
```

- model.ngcAPISecret and imagePullSecrets to communicate with NGC

```
model:
  nimCache: /model-store
  name: meta/llama3-8b-instruct # optional name of the model in the OpenAI API -- used in `helm test`
  ngcAPISecret: "ngc-api"
  ngcAPIKey: ""
  openaiPort: 8000
  labels: {}  # any extra labels desired on deployed pods
  jsonLogging: true
  logLevel: INFO
```

```
imagePullSecrets:
  - name: ngc-api  # change this to whatever your image pull secret should be
```

- **persistence.enabled** - Enables the use of persistent volumes.
- **persistence.storageClass** - Specifies the persistent volume storage class. If set to " -" , this disables dynamic provisioning. If left undefined or set to null, the cluster default storage provisioner is used.
- **persistence.size** - Specifies the size of the persistent volume claim (for example 40Gi).

```
persistence:
  enabled: true
  existingClaim: ""  # if using existingClaim, run only one replica or use a `ReadWriteMany` storage setup
  # Persistent Volume Storage Class
  # If defined, storageClassName: <storageClass>
  # If set to "-", storageClassName: "", which disables dynamic provisioning.
  # If undefined (the default) or set to null, no storageClassName spec is
  #   set, choosing the default provisioner.
  storageClass: "ontap-nfs"
  accessMode: ReadWriteOnce  # If using an NFS or similar setup, you can use `ReadWriteMany`
  stsPersistentVolumeClaimRetentionPolicy:
    whenDeleted: Retain
    whenScaled: Retain
  size: 500Gi  # size of claim in bytes (for example 8Gi)
```

service.type - Specifies the service type for the deployment.

```
service:
  type: ClusterIP
  openaiPort: 8000
```

Step 8. Once all the customizations are done, launch the chart:

```
helm install pkoppa-nim nim-llm-1.1.2.tgz -f values.yaml
```

Step 9. Wait for the pod to reach “Ready” status:

```
[admin@ocp-installer pkoppa-test]$ oc get pods
NAME               READY   STATUS    RESTARTS   AGE
pkoppa-nim-llm-0   1/1     Running   0          6d22h
```

In the previous example, the OpenAI compatible API endpoint is exposed on port 8000 through the Kubernetes service of the default type with no ingress since authentication is not handled by the NIM itself. The following commands assume the Llama 3 8B Instruct model was deployed. Adjust the “model” value in the request JSON body to use a different model.

Step 10. Use the following command to port-forward the service to your local machine to test inference:

```
oc port-forward service/pkoppa-nim-llm 8000:http-openai
```

Step 11. Run a request:

```
curl -X 'POST' \
  'http://localhost:8000/v1/chat/completions' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
"messages": [
{
"content": "You are a polite and respectful chatbot helping people plan a vacation.",
"role": "system"
},
{
"content": "What should I do for a 4 day vacation in Spain?",
"role": "user"
}
],
"model": "meta/llama3-8b-instruct",
"max_tokens": 16,
"top_p": 1,
"n": 1,
"stream": false,
"stop": "\n",
"frequency_penalty": 0.0
}'
```

Procedure 2. Deploy NVIDIA NeMo Retriever Embedding Microservice

This Helm Chart simplifies Embedding NIM deployment on Kubernetes. It aims to support deployment with a variety of possible cluster, GPU, and storage configurations.

More information on deploying NVIDA NIM for Text Embedding can be found here: https://catalog.ngc.nvidia.com/orgs/nim/teams/snowflake/helm-charts/text-embedding-nim

Step 1. Export the NGC API Key in the environment:

```
export NGC_CLI_API_KEY="<YOUR NGC API KEY>"
```

Step 2. Create a dedicated namespace called nrem:

```
oc create namespace nrem
```

Step 3. Change the namespace to the newly created one:

```
oc project nrem
```

Step 4. Use the following script below to create the expected secrets for this helm chart:

```
DOCKER_CONFIG='{"auths":{"nvcr.io":{"username":"$oauthtoken", "password":"'${NGC_API_KEY}'" }}}'
echo -n $DOCKER_CONFIG | base64 -w0
NGC_REGISTRY_PASSWORD=$(echo -n $DOCKER_CONFIG | base64 -w0 )

cat <<EOF > imagepull.yaml
apiVersion: v1
kind: Secret
metadata:
  name: nvcrimagepullsecret
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: ${NGC_REGISTRY_PASSWORD}
EOF

cat <<EOF > ngc-cli.yaml
apiVersion: v1
kind: Secret
metadata:
  name: ngc-api
type: Opaque
data:
  NGC_CLI_API_KEY: ${NGC_API_KEY}
EOF
```

Step 5. Apply the configuration:

```
oc apply -n nrem -f imagepull.yaml
oc apply -n nrem -f ngc-cli.yaml
```

Step 6. Fetch the helm chart from NGC.

```
helm fetch https://helm.ngc.nvidia.com/nim/snowflake/charts/text-embedding-nim-1.0.0.tgz
--username='$oauthtoken' --password=<YOUR NGC API KEY>
```

Step 7. Extract the content:

```
tar xvfz text-embedding-nim-1.0.0.tgz
```

Step 8. Change the directory to text-embedding-nim and customize values.yaml to install the chart. You can also install by passing the parameters as below.

Step 9. Install the helm chart for NIM Embedding:

```
helm upgrade --install \
  --namespace nrem \
  --set image.repository=nvcr.io/nim/snowflake/arctic-embed-l \
  --set image.tag=1.0.0 \
  --set imagePullSecret.password=$NGC_CLI_API_KEY \
  --set imagePullSecret.create=true \
  --set persistence.class=ontap-nfs \
  nemo-embedder \
  text-embedding-nim-1.0.0.tgz
```

Step 10. Check the status of the pods:

```
oc get pods -n nrem
```

The pod should be running and in a ready state within couple of minutes as shown below:

```
[admin@ocp-installer nemo-retriever]$ oc get pods
NAME                                         READY   STATUS    RESTARTS   AGE
nemo-embedder-nemo-retriever-embedding-ms-0  1/1     Running   0          6d23h
```

In the previous example the API endpoint is exposed on port 8080 through the Kubernetes service of the default type with no ingress since authentication is not handled by the NIM itself.

Step 11. Adjust the "model" value in the request JSON body to use a different model.

Step 12. Use the following command to port-forward the service to your local machine to test inference:

```
oc port-forward -n nrem service/nemo-embedder-nemo-retriever-embedding-ms 8080:8080
```

Step 13. Run a request:

```
curl -X 'POST' \
  'http://localhost:8080/v1/embeddings' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "input": "hello world",
    "model": "NV-Embed-QA",
    "input_type": "passage"
  }'
```

Procedure 3. Deploy Milvus vector store Helm Chart

Step 1. Create a new namespace for Milvus vector store:

```
oc create namespace vectorstore
```

Step 2. Change the namespace to the newly created one:

```
oc project vectorstore
```

Step 3. Add the Milvus Helm repository:

```
helm repo add milvus https://zilliztech.github.io/milvus-helm/
```

Step 4. Update charts locally:

```
helm repo update
```

Step 5. Create a file named custom_value.yaml, and update the parameters based on your environment before deploying Milvus:

Note: Milvus can be deployed in Standalone mode or Distributed mode.

Step 6. Standalone deployment of Milvus can be done with using this sample configuration to utilize GPUs:

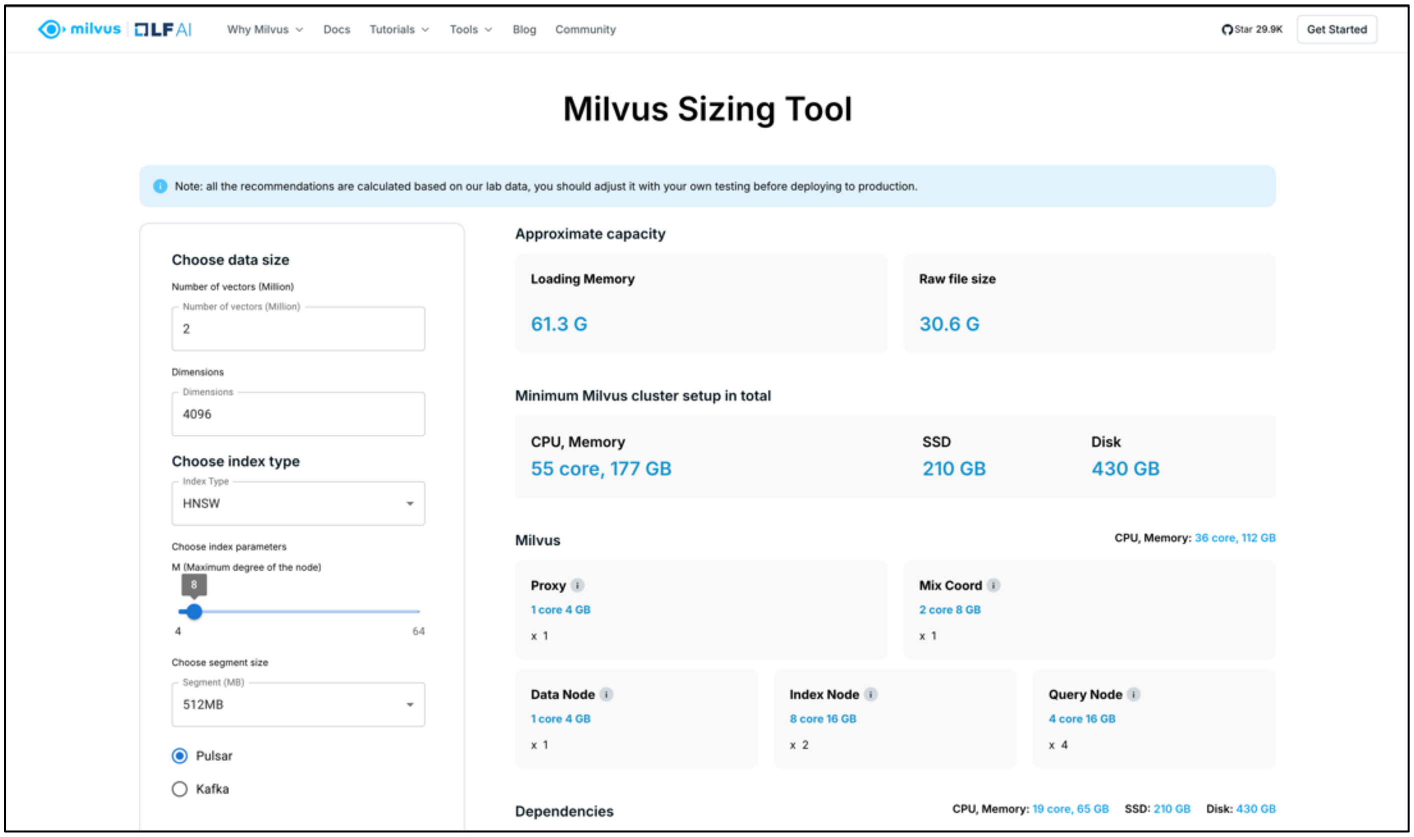
```
standalone:
  resources:
    requests:
      nvidia.com/gpu: "1"
    limits:
      nvidia.com/gpu: "1"
```

Step 7. A sample configuration for Milvus Distributed with Attu for UI is provided below:

```
attu:
  enabled: true
  name: attu
  image:
    repository: zilliz/attu
    tag: v2.2.8
    pullPolicy: IfNotPresent
  service:
    annotations: {}
    labels: {}
    type: ClusterIP
    port: 3000
    # loadBalancerIP: ""
  resources: {}
  ingress:
    enabled: false
    annotations: {}
    # Annotation example: set nginx ingress type
    # kubernetes.io/ingress.class: nginx
    labels: {}
    hosts:
      - milvus-attu.local
    tls: []
    #  - secretName: chart-attu-tls
    #    hosts:
    #      - milvus-attu.local


minio:
  resources:
    limits:
      cpu: 4
      memory: 16Gi
  persistence:
    storageClass:
    accessMode: ReadWriteOnce
    size: 900Gi
```

More information on installing Milvus can be found here: https://milvus.io/docs and go to **Get Started** > **Install Milvus**.

To generate the configuration with proper sizing for different components of Milvus, go to: https://milvus.io/tools/sizing. Provide the input such as the number of vectors, vector dimension etc. and download the helm chart configuration yaml manifest.



Milvus Sizing Tool

Note: all the recommendations are calculated based on our lab data, you should adjust it with your own testing before deploying to production.

Choose data size

Number of vectors (Million): 2

Dimensions: 4096

Choose index type

Index Type: HNSW

Choose index parameters

M (Maximum degree of the node): 8 (range 4–64)

Choose segment size

Segment (MB): 512MB

Pulsar [✓]

Kafka

Approximate capacity

Loading Memory: 61.3 G

Raw file size: 30.6 G

Minimum Milvus cluster setup in total

CPU, Memory: 55 core, 177 GB

SSD: 210 GB

Disk: 430 GB

Milvus — CPU, Memory: 36 core, 112 GB

Proxy: 1 core 4 GB x 1

Mix Coord: 2 core 8 GB x 1

Data Node: 1 core 4 GB x 1

Index Node: 8 core 16 GB x 2

Query Node: 4 core 16 GB x 4

Dependencies — CPU, Memory: 19 core, 65 GB SSD: 210 GB Disk: 430 GB

Step 8. Deploy the Milvus instance using helm chart:

```
helm install pkoppa-milvus  milvus/milvus -f custom_value.yaml
```

Step 9. Check the status of the pods once Milvus is deployed successfully in distributed mode:

```
[admin@ocp-installer milvus]$ oc -n vectorstore get pods
NAME                                     READY   STATUS      RESTARTS       AGE
pkoppa-milvus-attu-bbccfc786-58x5x       1/1     Running     0              21h
pkoppa-milvus-datanode-558f6548b7-dkb8c  1/1     Running     4 (21h ago)    21h
pkoppa-milvus-etcd-0                     1/1     Running     0              21h
pkoppa-milvus-etcd-1                     1/1     Running     0              21h
pkoppa-milvus-etcd-2                     1/1     Running     0              21h
pkoppa-milvus-indexnode-5c79dd47d4-95nqc 1/1     Running     3 (21h ago)    21h
pkoppa-milvus-minio-0                    1/1     Running     0              21h
pkoppa-milvus-minio-1                    1/1     Running     0              21h
pkoppa-milvus-minio-2                    1/1     Running     0              21h
pkoppa-milvus-minio-3                    1/1     Running     0              17h
pkoppa-milvus-mixcoord-d7d5485c-x98w6    1/1     Running     4 (21h ago)    21h
pkoppa-milvus-proxy-fbbb7b79-wkg29       1/1     Running     4 (21h ago)    21h
pkoppa-milvus-pulsar-bookie-0            1/1     Running     0              21h
pkoppa-milvus-pulsar-bookie-1            1/1     Running     0              21h
pkoppa-milvus-pulsar-bookie-2            1/1     Running     0              21h
pkoppa-milvus-pulsar-bookie-init-5tppx   0/1     Completed   0              21h
pkoppa-milvus-pulsar-broker-0            1/1     Running     0              21h
pkoppa-milvus-pulsar-proxy-0             1/1     Running     0              21h
pkoppa-milvus-pulsar-pulsar-init-xvvlv   0/1     Completed   0              21h
pkoppa-milvus-pulsar-recovery-0          1/1     Running     0              21h
pkoppa-milvus-pulsar-zookeeper-0         1/1     Running     0              21h
pkoppa-milvus-pulsar-zookeeper-1         1/1     Running     0              21h
pkoppa-milvus-pulsar-zookeeper-2         1/1     Running     0              21h
pkoppa-milvus-querynode-5cdb6f6886-2zbsg 1/1     Running     3 (21h ago)    21h
```

When Milvus is deployed in standalone mode, all pods should be running and in a ready state within couple of minutes.

```
NAME                                READY   STATUS    RESTARTS        AGE
milvus-etcd-0                       1/1     Running   0               5m34s
milvus-minio-76f9d647d5-44799       1/1     Running   0               5m34s
milvus-standalone-9ccf56df4-m4tpm   1/1     Running   3 (4m35s ago)   5m34
```

Step 10. Change the service type for Attu from ClusterIP to NodePort and save the file:

```
oc edit svc pkoppa-milvus -n vectorstore
```

Step 11. Confirm it has changed to NodePort:

```
admin@ocp-installer milvus]$ oc -n vectorstore get svc
NAME                               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
pkoppa-milvus                      ClusterIP   172.30.114.214   <none>        19530/TCP,9091/TCP
21h
pkoppa-milvus-attu                 NodePort    172.30.255.104   <none>        3000:30659/TCP
21h
pkoppa-milvus-datanode             ClusterIP   None             <none>        9091/TCP
21h
pkoppa-milvus-etcd                 ClusterIP   172.30.112.63    <none>        2379/TCP,2380/TCP
21h
pkoppa-milvus-etcd-headless        ClusterIP   None             <none>        2379/TCP,2380/TCP
21h
pkoppa-milvus-indexnode            ClusterIP   None             <none>        9091/TCP
21h
pkoppa-milvus-minio                ClusterIP   172.30.35.198    <none>        9000/TCP
21h
pkoppa-milvus-minio-svc            ClusterIP   None             <none>        9000/TCP
21h
pkoppa-milvus-mixcoord             ClusterIP   172.30.95.53     <none>        9091/TCP
21h
pkoppa-milvus-pulsar-bookie        ClusterIP   None             <none>        3181/TCP,8000/TCP
21h
pkoppa-milvus-pulsar-broker        ClusterIP   None             <none>        8080/TCP,6650/TCP
21h
pkoppa-milvus-pulsar-proxy         ClusterIP   172.30.77.96     <none>        8080/TCP,6650/TCP
21h
pkoppa-milvus-pulsar-recovery      ClusterIP   None             <none>        8000/TCP
21h
pkoppa-milvus-pulsar-zookeeper     ClusterIP   None             <none>
8000/TCP,2888/TCP,3888/TCP,2181/TCP   21h
pkoppa-milvus-querynode            ClusterIP   None             <none>        9091/TCP
21h
```

Procedure 4. Deploy RAG Application: Text QA Chatbot

Step 1. Create the example namespace:

```
oc create namespace canonical-rag
```

Step 2. Change the namespace to canonical-rag:

```
oc project canonical-rag
```

Step 3. To download the NIM container image, you must set an image pull secret, which is ngc-secret in the following example. To download model engines or weights from NGC, the chart requires a generic secret that has an NGC API key as a value stored in a key named NGC_API_KEY. The following example creates these two values:

```

oc create secret docker-registry ngc-secret --docker-server=nvcr.io --docker-username='$oauthtoken'
--docker-password=$NGC_API_KEY
```

```
oc create secret generic ngc-api --from-literal=NGC_API_KEY=$NGC_API_KEY
```

Step 4. Fetch the Helm chart from NGC:

```
helm fetch https://helm.ngc.nvidia.com/ohlfw0olaadg/ea-participants/charts/rag-app-text-chatbot-v24.06.tgz
--username='$oauthtoken' --password=<YOUR API KEY>
```

Step 5. Extract the content:

```
tar xvfz rag-app-text-chatbot-v24.06.tgz
```

Step 6. Change the directory to rag-app-text-chatbot and customize the values.yaml file.

Step 7. Update the imagePullSecret:

```
imagePullSecret:
  # Leave blank, if no imagePullSecret is needed.
  registry: "nvcr.io"
  name: "ngc-secret"
  # If set to false, the chart expects either a imagePullSecret
  # with the name configured above to be present on the cluster or that no
  # credentials are needed.
  create: true
  username: '$oauthtoken'
  password: <YOUR NGC API KEY>
```

Step 8. Update the **query.env** with the LLM server URL, embedding server URL, model names and vector store URL:

```
query:
  image: nvcr.io/ohlfw0olaadg/ea-participants/rag-application-text-chatbot:24.06
  name: chain-server-canonical-rag
  replicas: 1
  nodeSelector: {}
  tolerations: {}
  affinity: {}
  env:
      APP_VECTORSTORE_URL: "http://pkoppa-milvus.vectorstore:19530"
      APP_VECTORSTORE_NAME: "milvus"
      COLLECTION_NAME: canonical_rag
      APP_LLM_SERVERURL: "pkoppa-nim-llm.nim-llm:8000"
      APP_LLM_MODELNAME: meta/llama3-8b-instruct
      APP_LLM_MODELENGINE: nvidia-ai-endpoints
      APP_EMBEDDINGS_SERVERURL: "nemo-embedder-nemo-retriever-embedding-ms.text-embedding-nim:8080"
      APP_EMBEDDINGS_MODELNAME: NV-Embed-QA
      APP_EMBEDDINGS_MODELENGINE: nvidia-ai-endpoints
      APP_TEXTSPLITTER_CHUNKSIZE: 506
      APP_TEXTSPLITTER_CHUNKOVERLAP: 200
```

Step 9. To get Milvus vector store URL, look for the service running in the vectorstore namespace. Repeat it for NIM for LLM and embedding server URL in their corresponding namespace:

```
[admin@ocp-installer rag-app-text-chatbot]$ oc get svc -n vectorstore
NAME                             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
pkoppa-milvus                    ClusterIP   172.30.114.214   <none>        19530/TCP,9091/TCP
22h
pkoppa-milvus-attu               NodePort    172.30.255.104   <none>        3000:30659/TCP
22h
pkoppa-milvus-datanode           ClusterIP   None             <none>        9091/TCP
22h
pkoppa-milvus-etcd               ClusterIP   172.30.112.63    <none>        2379/TCP,2380/TCP
22h
pkoppa-milvus-etcd-headless      ClusterIP   None             <none>        2379/TCP,2380/TCP
22h
```

```
pkoppa-milvus-indexnode          ClusterIP   None            <none>        9091/TCP
22h
pkoppa-milvus-minio              ClusterIP   172.30.35.198   <none>        9000/TCP
22h
pkoppa-milvus-minio-svc          ClusterIP   None            <none>        9000/TCP
22h
pkoppa-milvus-mixcoord           ClusterIP   172.30.95.53    <none>        9091/TCP
22h
pkoppa-milvus-pulsar-bookie      ClusterIP   None            <none>        3181/TCP,8000/TCP
22h
pkoppa-milvus-pulsar-broker      ClusterIP   None            <none>        8080/TCP,6650/TCP
22h
pkoppa-milvus-pulsar-proxy       ClusterIP   172.30.77.96    <none>        8080/TCP,6650/TCP
22h
pkoppa-milvus-pulsar-recovery    ClusterIP   None            <none>        8000/TCP
22h
pkoppa-milvus-pulsar-zookeeper   ClusterIP   None            <none>
8000/TCP,2888/TCP,3888/TCP,2181/TCP   22h
pkoppa-milvus-querynode          ClusterIP   None            <none>        9091/TCP
22h
```

Step 10. Once helm chart configurations are done, create the canonical rag pipeline instance, and start the services:

```
helm install canonical-rag
```

Step 11. Verify the pods are running and ready:

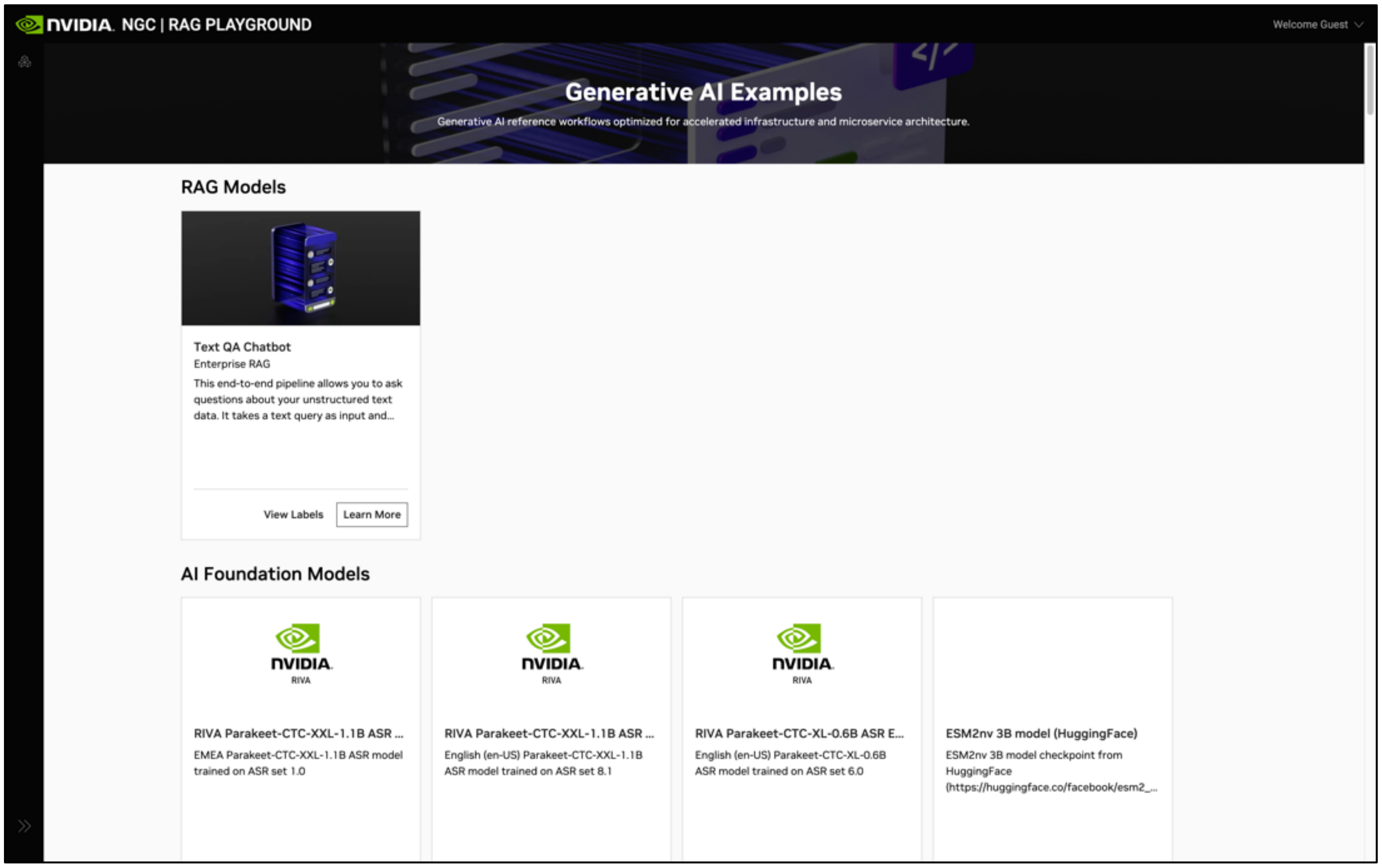
```
[admin@ocp-installer rag-app-text-chatbot]$ oc -n canonical-rag get pods
NAME                               READY   STATUS    RESTARTS   AGE
chain-server-5446654586-rt489      1/1     Running   0          21h
rag-playground-54b9d55b7c-qvhts    1/1     Running   0          21h
```

Step 12. Access the application using port-forwarding:

```
oc port-forward service/rag-playground -n canonical-rag 30001:3001
```

Step 13. Open browser and access the rag-playground UI using below URL:

```
http://localhost:30001/orgs/nvidia/models/text-qa-chatbot
```


NVIDIA. NGC | RAG PLAYGROUND
Welcome Guest
Generative AI Examples
Generative AI reference workflows optimized for accelerated infrastructure and microservice architecture.
RAG Models
View Labels
Learn More
AI Foundation Models
NVIDIA RIVA
NVIDIA RIVA
NVIDIA RIVA

Text QA Chatbot
Enterprise RAG
This end-to-end pipeline allows you to ask questions about your unstructured text data. It takes a text query as input and...

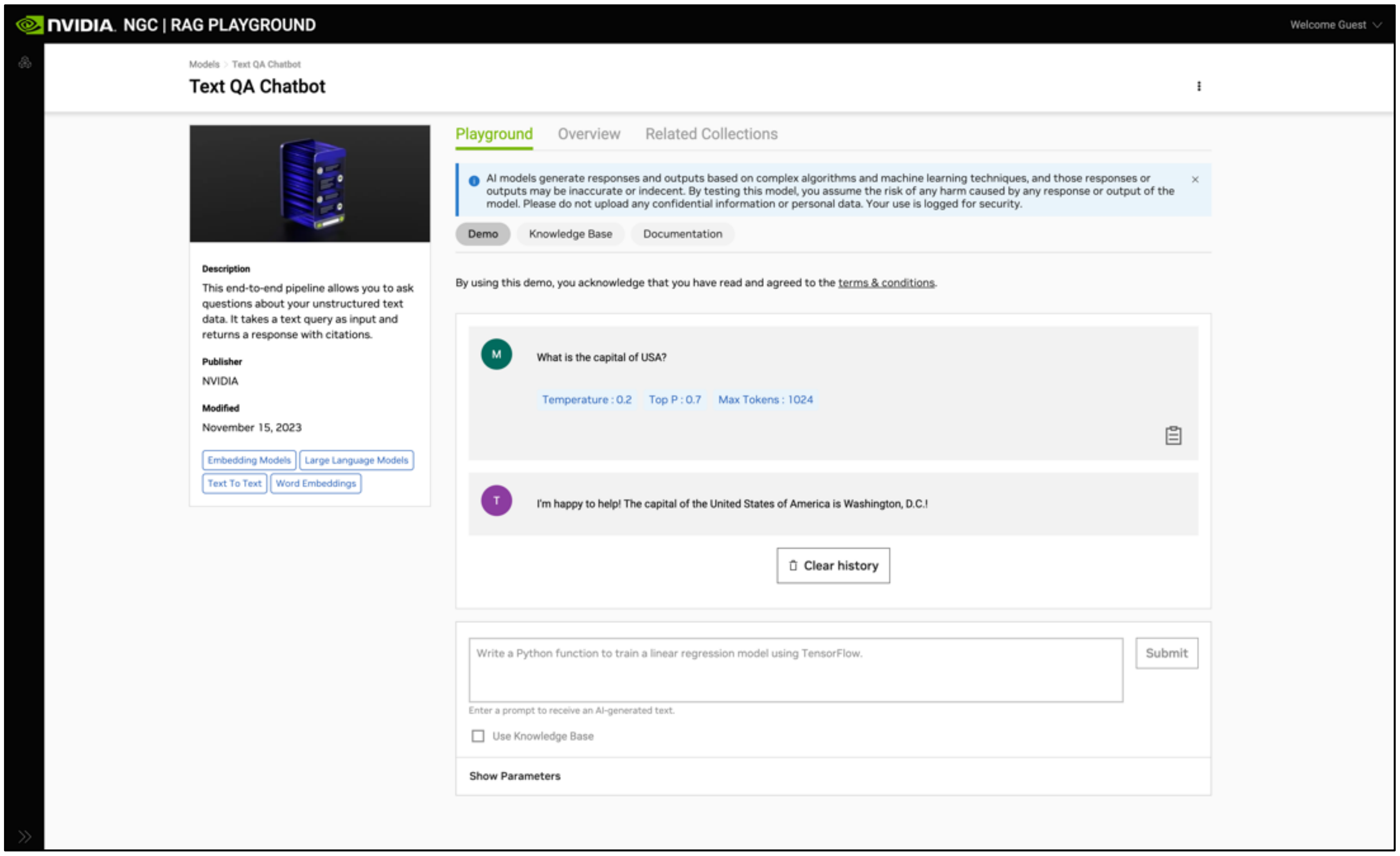
RIVA Parakeet-CTC-XXL-1.1B ASR ...
EMEA Parakeet-CTC-XXL-1.1B ASR model trained on ASR set 1.0

RIVA Parakeet-CTC-XXL-1.1B ASR ...
English (en-US) Parakeet-CTC-XXL-1.1B ASR model trained on ASR set 8.1

RIVA Parakeet-CTC-XL-0.6B ASR E...
English (en-US) Parakeet-CTC-XL-0.6B ASR model trained on ASR set 6.0

ESM2nv 3B model (HuggingFace)
ESM2nv 3B model checkpoint from HuggingFace (https://huggingface.co/facebook/esm2_...

Step 14. Click **Text QA Chatbot** application and ask a question and make sure you get response.



Text QA Chatbot

Description
This end-to-end pipeline allows you to ask questions about your unstructured text data. It takes a text query as input and returns a response with citations.

Publisher
NVIDIA

Modified
November 15, 2023

AI models generate responses and outputs based on complex algorithms and machine learning techniques, and those responses or outputs may be inaccurate or indecent. By testing this model, you assume the risk of any harm caused by any response or output of the model. Please do not upload any confidential information or personal data. Your use is logged for security.

By using this demo, you acknowledge that you have read and agreed to the terms & conditions.

M

What is the capital of USA?

Temperature : 0.2 Top P : 0.7 Max Tokens : 1024

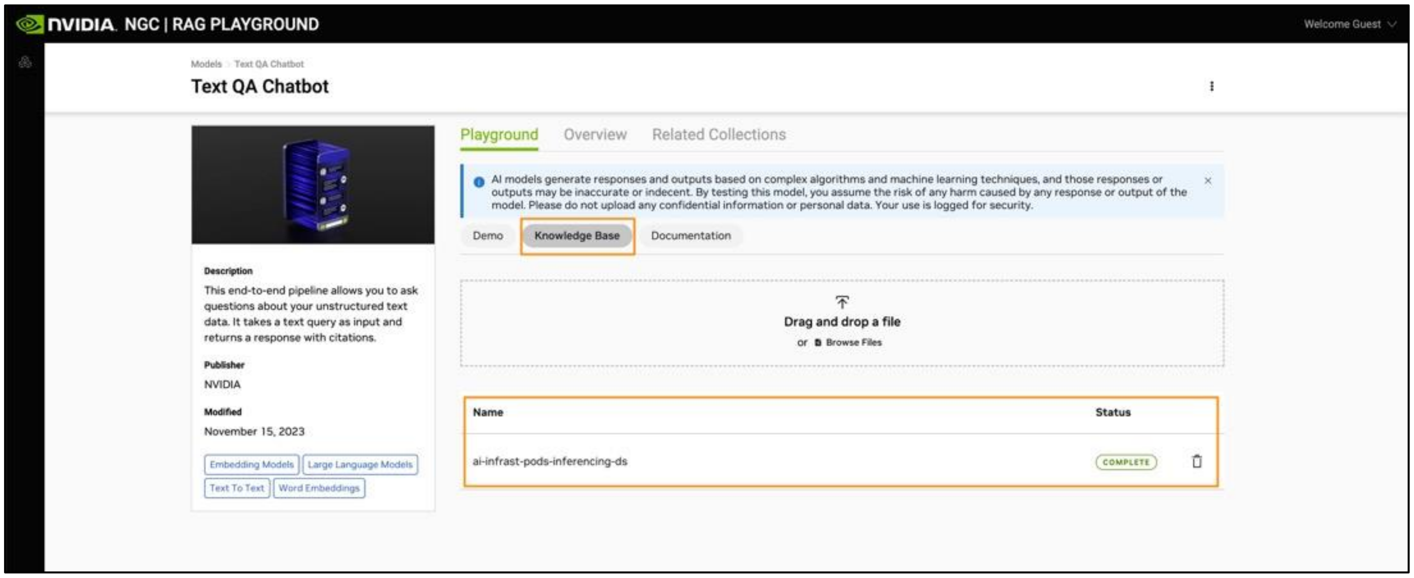
T

I'm happy to help! The capital of the United States of America is Washington, D.C.!

Write a Python function to train a linear regression model using TensorFlow.

Enter a prompt to receive an AI-generated text.

Step 15. Upload a document to Knowledge Base and make sure it's updated.



Text QA Chatbot

Description
This end-to-end pipeline allows you to ask questions about your unstructured text data. It takes a text query as input and returns a response with citations.

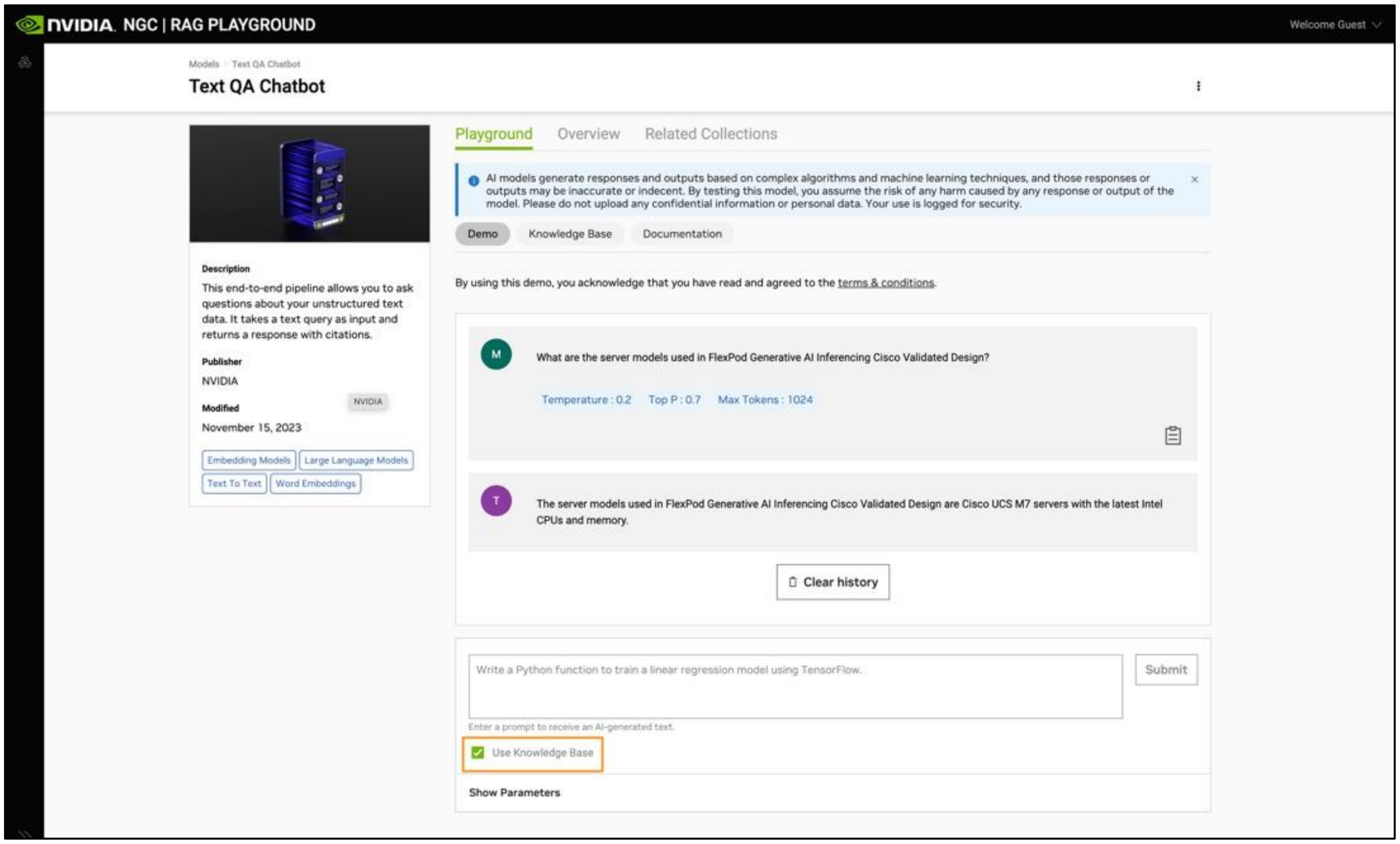
Publisher
NVIDIA

Modified
November 15, 2023

AI models generate responses and outputs based on complex algorithms and machine learning techniques, and those responses or outputs may be inaccurate or indecent. By testing this model, you assume the risk of any harm caused by any response or output of the model. Please do not upload any confidential information or personal data. Your use is logged for security.

Name	Status
ai-infrast-pods-inferencing-ds	COMPLETE

Step 16. Ask a question. Click **Use Knowledge Base** so that LLM refers to uploaded document to provide accurate answer.



Text QA Chatbot

AI models generate responses and outputs based on complex algorithms and machine learning techniques, and those responses or outputs may be inaccurate or indecent. By testing this model, you assume the risk of any harm caused by any response or output of the model. Please do not upload any confidential information or personal data. Your use is logged for security.

By using this demo, you acknowledge that you have read and agreed to the terms & conditions.

Description
This end-to-end pipeline allows you to ask questions about your unstructured text data. It takes a text query as input and returns a response with citations.

Publisher
NVIDIA

Modified
November 15, 2023

M: What are the server models used in FlexPod Generative AI Inferencing Cisco Validated Design?

Temperature : 0.2 Top P : 0.7 Max Tokens : 1024

T: The server models used in FlexPod Generative AI Inferencing Cisco Validated Design are Cisco UCS M7 servers with the latest Intel CPUs and memory.

Write a Python function to train a linear regression model using TensorFlow.

Enter a prompt to receive an AI-generated text.

✓ Use Knowledge Base

Step 17. Make sure vector database shows the **canonical_rag** collection.

Step 18. Get the compute node IP/DNS for one of the node:

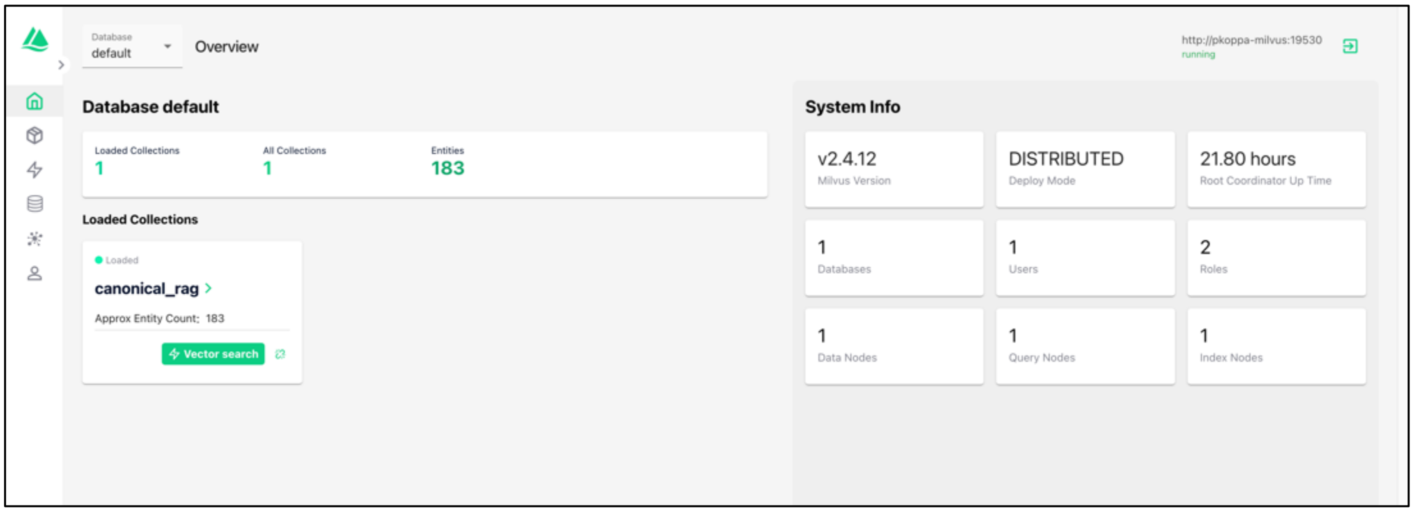
```
[admin@ocp-installer rag-app-text-chatbot]$ oc get nodes -o wide
NAME                                      STATUS   ROLES                   AGE   VERSION            INTERNAL-IP
EXTERNAL-IP   OS-IMAGE                                                     KERNEL-VERSION
CONTAINER-RUNTIME
ocp-master0.ocp.flexpodb4.cisco.com   Ready    control-plane,master   28d   v1.29.7+4510e9c   10.102.2.211
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
ocp-master1.ocp.flexpodb4.cisco.com   Ready    control-plane,master   28d   v1.29.7+4510e9c   10.102.2.212
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
ocp-master2.ocp.flexpodb4.cisco.com   Ready    control-plane,master   28d   v1.29.7+4510e9c   10.102.2.213
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
ocp-worker0.ocp.flexpodb4.cisco.com   Ready    worker                 15d   v1.29.7+4510e9c   10.102.2.214
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
ocp-worker1.ocp.flexpodb4.cisco.com   Ready    worker                 28d   v1.29.7+4510e9c   10.102.2.215
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
ocp-worker2.ocp.flexpodb4.cisco.com   Ready    worker                 15d   v1.29.7+4510e9c   10.102.2.216
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
```

Step 19. Get the node port for Attu:

```
[admin@ocp-installer rag-app-text-chatbot]$ oc -n vectorstore get svc pkoppa-milvus-attu
NAME                 TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
pkoppa-milvus-attu   NodePort   172.30.255.104   <none>        3000:30659/TCP   21h
```

Step 20. Open browser and access the Attu UI using <Any of the Node IP/DNS:Attu Node Port>:

```
http://10.102.2.214:30659/
```



Database default

Loaded Collections	All Collections	Entities
1	1	183

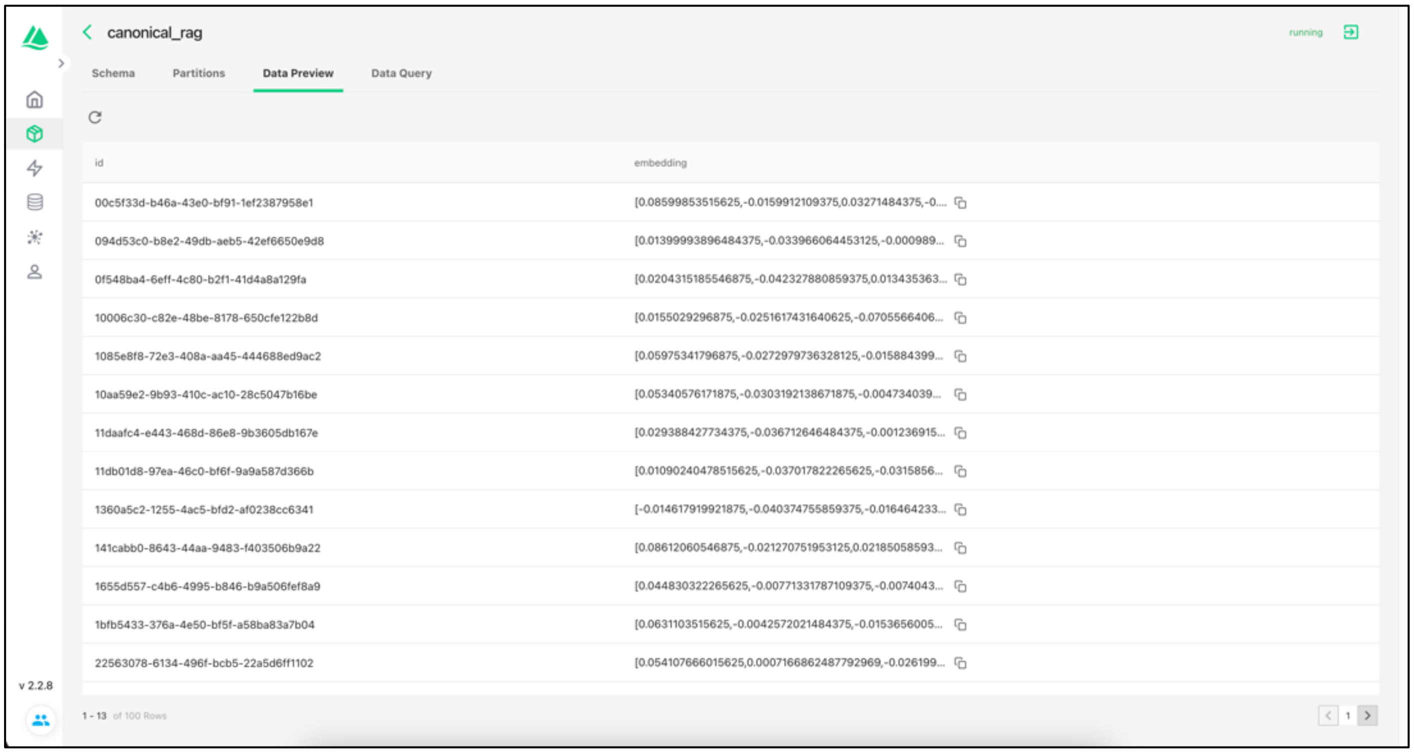
Loaded Collections

Loaded
canonical_rag
Approx Entity Count: 183

System Info

v2.4.12 Milvus Version	DISTRIBUTED Deploy Mode	21.80 hours Root Coordinator Up Time
1 Databases	1 Users	2 Roles
1 Data Nodes	1 Query Nodes	1 Index Nodes

Step 21. Click the **canonical_rag** collection and navigate to Data Preview and observe that embeddings are stored in Milvus.



id	embedding
00c5f33d-b46a-43e0-bf91-1ef2387958e1	[0.08599853515625,-0.0159912109375,0.03271484375,-0....
094d53c0-b8e2-49db-aeb5-42ef6650e9d8	[0.01399993896484375,-0.033966064453125,-0.000989...
0f548ba4-6eff-4c80-b2f1-41d4a8a129fa	[0.0204315185546875,-0.042327880859375,0.013435363...
10006c30-c82e-48be-8178-650cfe122b8d	[0.0155029296875,-0.0251617431640625,-0.0705566406...
1085e8f8-72e3-408a-aa45-444688ed9ac2	[0.05975341796875,-0.0272979736328125,-0.015884399...
10aa59e2-9b93-410c-ac10-28c5047b16be	[0.05340576171875,-0.0303192138671875,-0.004734039...
11daafc4-e443-468d-86e8-9b3605db167e	[0.029388427734375,-0.036712646484375,-0.001236915...
11db01d8-97ea-46c0-bf6f-9a9a587d366b	[0.01090240478515625,-0.03701782265625,-0.0315856...
1360a5c2-1255-4ac5-bfd2-af0238cc6341	[-0.014617919921875,-0.040374755859375,-0.016464233...
141cabb0-8643-44aa-9483-f403506b9a22	[0.08612060546875,-0.021270751953125,0.02185058593...
1655d557-c4b6-4995-b846-b9a506fef8a9	[0.044830322265625,-0.00771331787109375,-0.0074043...
1bfb5433-376a-4e50-bf5f-a58ba83a7b04	[0.0631103515625,-0.0042572021484375,-0.0153656005...
22563078-6134-496f-bcb5-22a5d6ff1102	[0.054107666015625,0.0007166862487792969,-0.026199...

1 - 13 of 100 Rows

Multi-turn RAG

This example showcases multi turn use case in a RAG pipeline. It stores the conversation history and knowledge base in Milvus and retrieves them at runtime to understand contextual queries. It uses NeMo Inference Micro-services to communicate with the embedding model and large language model. The example supports ingestion of PDF, .txt files. The docs are ingested in a dedicated document vector store. The prompt for the example is currently tuned to act as a document chat bot. For maintaining the conversation history, we store the previous query of user and its generated answer as a text entry in a different dedicated vector store for conversation history. Both these vector stores are part of a Langchain LCEL (LangChain Expression Language) chain as Langchain Retrievers. When the chain is invoked with a query, it's passed through both the retrievers. The retriever retrieves context from the document vector store and the closest matching conversation history from conversation history vector store and the chunks are added into the LLM prompt as part of the chain.

LLM Model	Embedding	Framework	Document Type	Vector Database	Model deployment platform
meta/llama3-8b-instruct	snowflake-arctic-embed-l	Langchain Expression Language	PDF/Text	Milvus	On Prem

Prerequisites

- You have the NGC CLI available on your client machine. You can download the CLI from https://ngc.nvidia.com/setup/installers/cli.
- NGC setup - https://org.ngc.nvidia.com/setup
- You have a default storage class available in the OpenShift cluster for PVC provisioning.

Note: Deploying NVIDIA NIM for LLM, Deploying NVIDIA NeMo Retriever Embedding Microservice, and Deploying Milvus Vector store Helm Chart as detailed in section [Canonical RAG](#).

Procedure 1. Deploy RAG Application: Multi-turn Agent

Step 1. Create the example namespace:

```
oc create namespace multiturn-rag
```

Step 2. Change the namespace to multiturn-rag:

```
oc project multiturn-rag
```

Step 3. To download the NIM container image, you must set an image pull secret, which is ngc-secret in the following example. To download model engines or weights from NGC, the chart requires a generic secret that has an NGC API key as a value stored in a key named NGC_API_KEY. The following example creates these two values:

```
oc create secret docker-registry ngc-secret-multi-turn --docker-server=nvcr.io --docker-username='$oauthtoken'
--docker-password=$NGC_API_KEY


oc create secret generic ngc-api --from-literal=NGC_API_KEY=$NGC_API_KEY
```

Step 4. Fetch the Helm chart from NGC:

```
helm fetch https://helm.ngc.nvidia.com/nvidia/aiworkflows/charts/rag-app-multiturn-chatbot-24.08.tgz
--username='$oauthtoken' --password=<YOUR NGC API KEY>
```

Step 5. Extract the content:

```
tar xvfz rag-app-multiturn-chatbot-v24.08.tgz
```

Step 6. Change the directory to rag-app-multiturn-chatbot and update the helm chart's values.yaml manifest.

Step 7. Update the imagePullSecret:

```
imagePullSecret:
  # Leave blank, if no imagePullSecret is needed.
  registry: "nvcr.io"
  name: "ngc-secret-multi-turn"
  # If set to false, the chart expects either a imagePullSecret
  # with the name configured above to be present on the cluster or that no
  # credentials are needed.
  create: true
  username: '$oauthtoken'
  password: <NGC_API_KEY>
```

Step 8. Update the **query.env** with the LLM server URL, embedding server URL, model names and vector store URL:

```
query:
  image: nvcr.io/ohlfw0olaadg/ea-participants/rag-application-multiturn-chatbot:24.06
  name: chain-server-multi-turn
  replicas: 1
  nodeSelector: {}
  tolerations: {}
  affinity: {}
  env:
      APP_VECTORSTORE_URL: "http://pkoppa-milvus.vectorstore:19530"
      APP_VECTORSTORE_NAME: "milvus"
      APP_LLM_SERVERURL: "pkoppa-nim-llm.nim-llm:8000"
      APP_LLM_MODELNAME: meta/llama3-8b-instruct
      APP_LLM_MODELENGINE: nvidia-ai-endpoints
      APP_EMBEDDINGS_SERVERURL: "nemo-embedder-nemo-retriever-embedding-ms.text-embedding-nim:8080"
      APP_EMBEDDINGS_MODELNAME: NV-Embed-QA
      APP_EMBEDDINGS_MODELENGINE: nvidia-ai-endpoints
      COLLECTION_NAME: multi_turn_rag
      APP_RETRIEVER_TOPK: 2
      APP_RETRIEVER_SCORETHRESHOLD: 0.25
      APP_TEXTSPLITTER_CHUNKSIZE: 506
      APP_TEXTSPLITTER_CHUNKOVERLAP: 200
```

Step 9. To get vector store URL, look for the service running in the vectorstore namespace. Repeat it for NIM for LLM and embedding server URL in their corresponding namespace:

```
[admin@ocp-installer rag-app-text-chatbot]$ oc get svc -n vectorstore
NAME                             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
pkoppa-milvus                    ClusterIP   172.30.114.214   <none>        19530/TCP,9091/TCP
22h
pkoppa-milvus-attu               NodePort    172.30.255.104   <none>        3000:30659/TCP
22h
pkoppa-milvus-datanode           ClusterIP   None             <none>        9091/TCP
22h
pkoppa-milvus-etcd               ClusterIP   172.30.112.63    <none>        2379/TCP,2380/TCP
22h
pkoppa-milvus-etcd-headless      ClusterIP   None             <none>        2379/TCP,2380/TCP
22h
pkoppa-milvus-indexnode          ClusterIP   None             <none>        9091/TCP
22h
pkoppa-milvus-minio              ClusterIP   172.30.35.198    <none>        9000/TCP
22h
pkoppa-milvus-minio-svc          ClusterIP   None             <none>        9000/TCP
22h
pkoppa-milvus-mixcoord           ClusterIP   172.30.95.53     <none>        9091/TCP
22h
pkoppa-milvus-pulsar-bookie      ClusterIP   None             <none>        3181/TCP,8000/TCP
22h
```

```
pkoppa-milvus-pulsar-broker      ClusterIP   None            <none>        8080/TCP,6650/TCP
22h
pkoppa-milvus-pulsar-proxy       ClusterIP   172.30.77.96    <none>        8080/TCP,6650/TCP
22h
pkoppa-milvus-pulsar-recovery    ClusterIP   None            <none>        8000/TCP
22h
pkoppa-milvus-pulsar-zookeeper   ClusterIP   None            <none>
8000/TCP,2888/TCP,3888/TCP,2181/TCP   22h
pkoppa-milvus-querynode          ClusterIP   None            <none>        9091/TCP
22h
```

Step 10. Once helm chart configurations are done, create the multi turn rag pipeline instance and start the services:

```
helm install multiturn-rag .
```

Step 11. Verify the pods are running and ready:

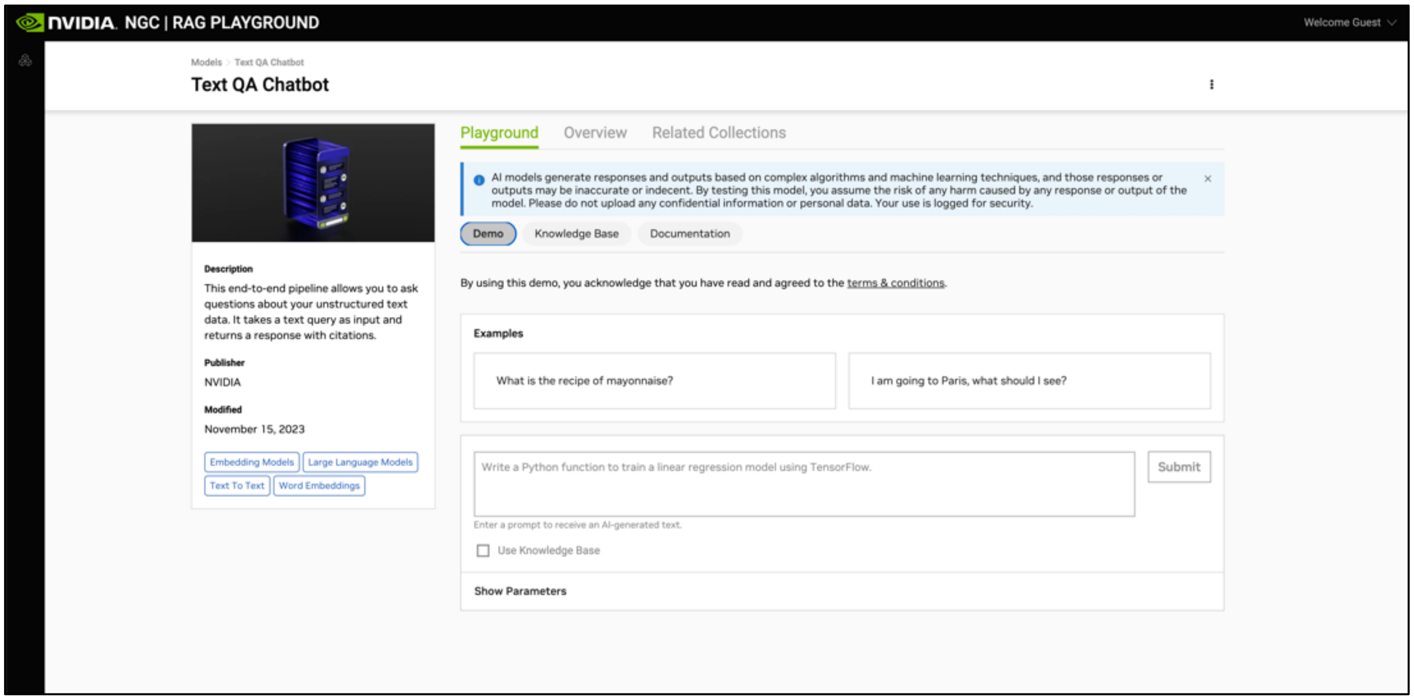
```
[admin@ocp-installer rag-app-multiturn-chatbot]$ oc get pods -n multiturn-rag
NAME                                             READY   STATUS    RESTARTS   AGE
chain-server-multi-turn-78c575d6db-mxj5t         1/1     Running   0          5s
rag-playground-multiturn-rag-5cbdc574d6-v5x67    1/1     Running   0          5s
```

Step 12. Get the node port for rag-playground-multiturn-rag service using below command:

```
[admin@ocp-installer rag-app-multiturn-chatbot]$ oc get svc -n multiturn-rag
NAME                           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
chain-server-multi-turn        ClusterIP   172.30.4.16      <none>        8082/TCP         31s
rag-playground-multiturn-rag   NodePort    172.30.119.148   <none>        3001:31002/TCP   31s
```

Step 13. Open browser and access the rag-playground UI using one of the node IP/DNS:node port:

```
http://localhost:30001/orgs/nvidia/models/text-qa-chatbot
```



AI models generate responses and outputs based on complex algorithms and machine learning techniques, and those responses or outputs may be inaccurate or indecent. By testing this model, you assume the risk of any harm caused by any response or output of the model. Please do not upload any confidential information or personal data. Your use is logged for security.

By using this demo, you acknowledge that you have read and agreed to the terms & conditions.

What is the recipe of mayonnaise?

I am going to Paris, what should I see?

Write a Python function to train a linear regression model using TensorFlow.

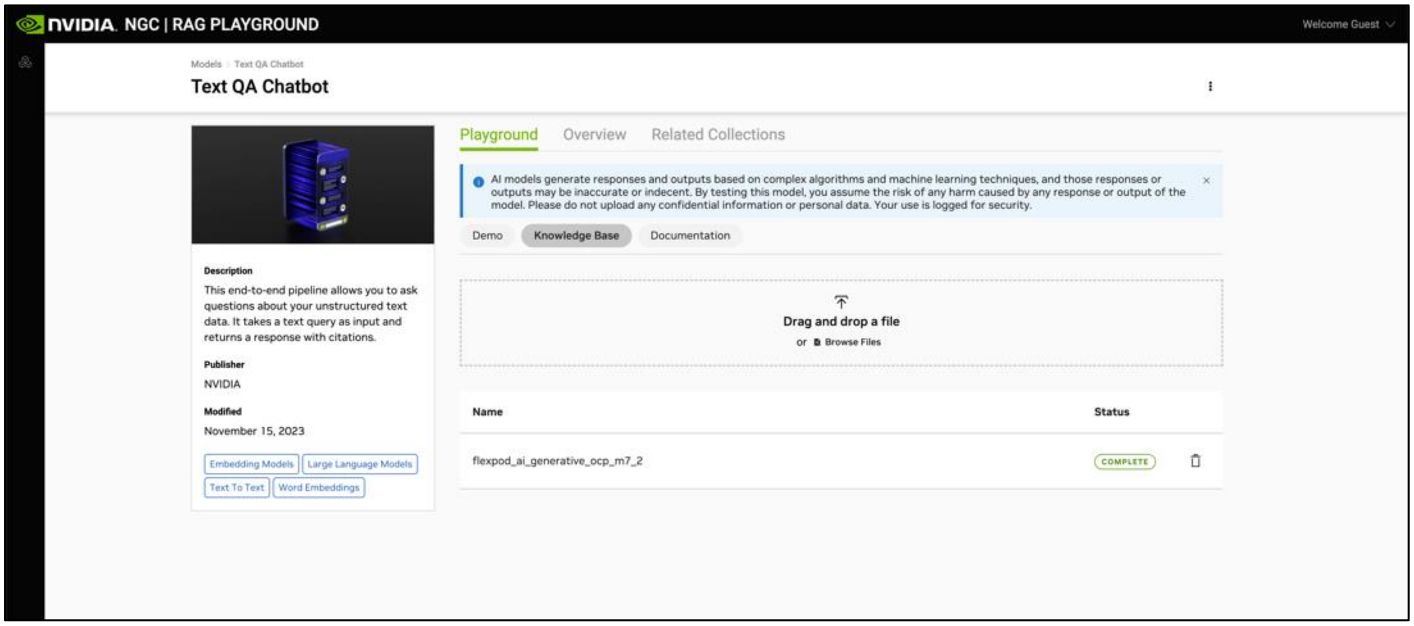
Enter a prompt to receive an AI-generated text.

Description
This end-to-end pipeline allows you to ask questions about your unstructured text data. It takes a text query as input and returns a response with citations.

Publisher
NVIDIA

Modified
November 15, 2023

Step 14. Upload a document and make sure upload goes successful:


NVIDIA. NGC | RAG PLAYGROUND

Models › Text QA Chatbot

Text QA Chatbot

Description

This end-to-end pipeline allows you to ask questions about your unstructured text data. It takes a text query as input and returns a response with citations.

Publisher

NVIDIA

Modified

November 15, 2023

AI models generate responses and outputs based on complex algorithms and machine learning techniques, and those responses or outputs may be inaccurate or indecent. By testing this model, you assume the risk of any harm caused by any response or output of the model. Please do not upload any confidential information or personal data. Your use is logged for security.

Name	Status
flexpod_ai_generative_ocp_m7_2	COMPLETE

Step 15. Make sure vector database shows the **multi_turn_rag** collection.

Step 16. Get the compute node IP/DNS for one of the node:

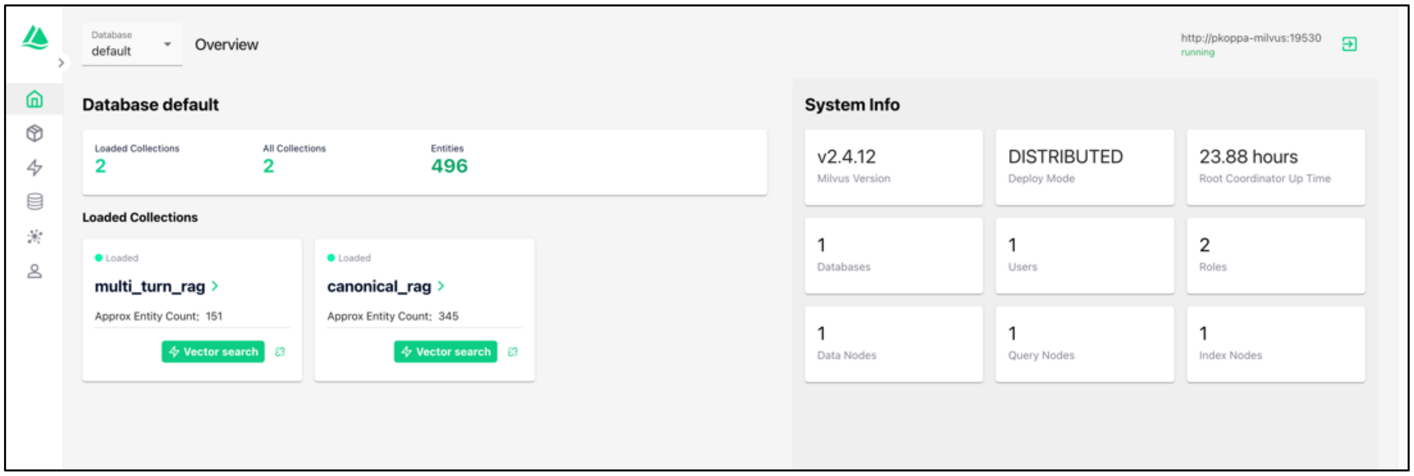
```
[admin@ocp-installer rag-app-text-chatbot]$ oc get nodes -o wide
NAME                                STATUS   ROLES                  AGE   VERSION           INTERNAL-IP
EXTERNAL-IP   OS-IMAGE                                              KERNEL-VERSION
CONTAINER-RUNTIME
ocp-master0.ocp.flexpodb4.cisco.com   Ready    control-plane,master   28d   v1.29.7+4510e9c   10.102.2.211
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
ocp-master1.ocp.flexpodb4.cisco.com   Ready    control-plane,master   28d   v1.29.7+4510e9c   10.102.2.212
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
ocp-master2.ocp.flexpodb4.cisco.com   Ready    control-plane,master   28d   v1.29.7+4510e9c   10.102.2.213
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
ocp-worker0.ocp.flexpodb4.cisco.com   Ready    worker                 15d   v1.29.7+4510e9c   10.102.2.214
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
ocp-worker1.ocp.flexpodb4.cisco.com   Ready    worker                 28d   v1.29.7+4510e9c   10.102.2.215
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
ocp-worker2.ocp.flexpodb4.cisco.com   Ready    worker                 15d   v1.29.7+4510e9c   10.102.2.216
<none>        Red Hat Enterprise Linux CoreOS 416.94.202408260940-0   5.14.0-427.33.1.el9_4.x86_64
cri-o://1.29.7-5.rhaos4.16.gitb130ec5.el9
```

Step 17. Get the node port for Attu:

```
[admin@ocp-installer rag-app-text-chatbot]$ oc -n vectorstore get svc pkoppa-milvus-attu
NAME                 TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
pkoppa-milvus-attu   NodePort   172.30.255.104   <none>        3000:30659/TCP   21h
```

Step 18. Open browser and access the Attu UI using <Any of the Node IP/DNS:Attu Node Port>:

```
http://10.102.2.214:30659/
```



Database default

Overview

http://pkoppa-milvus:19530
running

Database default

Loaded Collections	All Collections	Entities
2	2	496

Loaded Collections

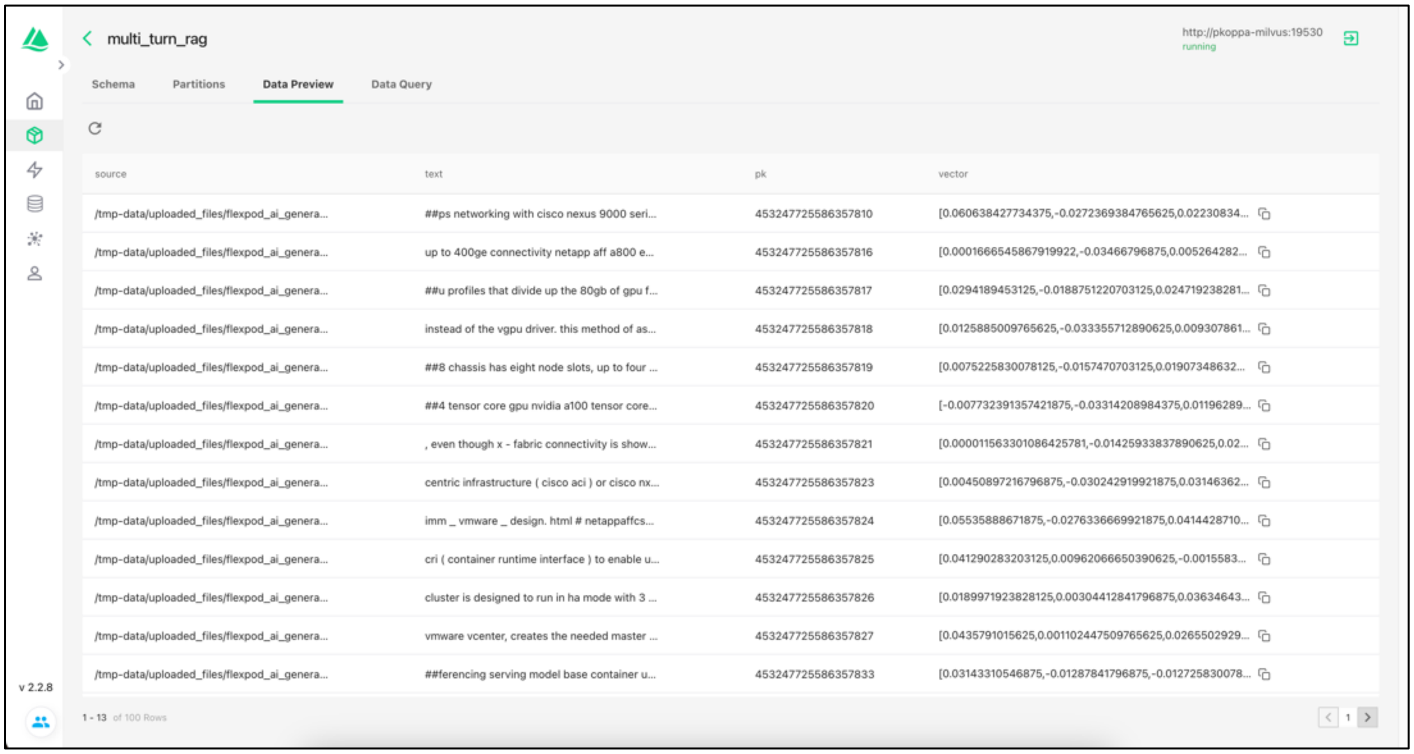
Loaded
multi_turn_rag
Approx Entity Count: 151

Loaded
canonical_rag
Approx Entity Count: 345

System Info

v2.4.12 Milvus Version	DISTRIBUTED Deploy Mode	23.88 hours Root Coordinator Up Time
1 Databases	1 Users	2 Roles
1 Data Nodes	1 Query Nodes	1 Index Nodes

Step 19. Click the **multi_turn_rag** collection and navigate to **Data Preview** and observe that embeddings are stored in Milvus.



multi_turn_rag

http://pkoppa-milvus:19530
running

source	text	pk	vector
/tmp-data/uploaded_files/flexpod_ai_genera...	##ps networking with cisco nexus 9000 seri...	453247725586357810	[0.060638427734375,-0.0272369384765625,0.02230834...
/tmp-data/uploaded_files/flexpod_ai_genera...	up to 400ge connectivity netapp aff a800 e...	453247725586357816	[0.0001666545867919922,-0.03466796875,0.005264282...
/tmp-data/uploaded_files/flexpod_ai_genera...	##u profiles that divide up the 80gb of gpu f...	453247725586357817	[0.0294189453125,-0.0188751220703125,0.024719238281...
/tmp-data/uploaded_files/flexpod_ai_genera...	instead of the vgpu driver. this method of as...	453247725586357818	[0.0125885009765625,-0.033355712890625,0.009307861...
/tmp-data/uploaded_files/flexpod_ai_genera...	##8 chassis has eight node slots, up to four ...	453247725586357819	[0.0075225830078125,-0.0157470703125,0.01907348632...
/tmp-data/uploaded_files/flexpod_ai_genera...	##4 tensor core gpu nvidia a100 tensor core...	453247725586357820	[-0.007732391357421875,-0.03314208984375,0.01196289...
/tmp-data/uploaded_files/flexpod_ai_genera...	, even though x - fabric connectivity is show...	453247725586357821	[0.000011563301086425781,-0.014259338378906250,0.02...
/tmp-data/uploaded_files/flexpod_ai_genera...	centric infrastructure (cisco aci) or cisco nx...	453247725586357823	[0.0045089721679687 5,-0.030242919921875,0.03146362...
/tmp-data/uploaded_files/flexpod_ai_genera...	imm _ vmware _ design. html # netappaffcs...	453247725586357824	[0.05535888671875,-0.0276336669921875,0.0414428710...
/tmp-data/uploaded_files/flexpod_ai_genera...	cri (container runtime interface) to enable u...	453247725586357825	[0.041290283203125,0.0096206665039062 5,-0.0015583...
/tmp-data/uploaded_files/flexpod_ai_genera...	cluster is designed to run in ha mode with 3 ...	453247725586357826	[0.0189971923828125,0.0030441284179687 5,0.03634643...
/tmp-data/uploaded_files/flexpod_ai_genera...	vmware vcenter, creates the needed master ...	453247725586357827	[0.0435791015625,0.0011024475097656 25,0.0265502929...
/tmp-data/uploaded_files/flexpod_ai_genera...	##ferencing serving model base container u...	453247725586357833	[0.03143310546875,-0.01287841796875,-0.012725830078...

1 - 13 of 100 Rows

Benchmarking, Evaluation, and Sizing

This chapter contains the following:

This section examines the performance benchmarking of NIM for LLM, RAG Evaluation using RAGAS framework, Sizing guidance for GPU and model selection and vector database benchmarking.

NIM for LLM Benchmarking

Benchmarking the deployment of Large Language Models (LLMs) involves understanding key metrics such as inference latency and throughput, which are crucial for evaluating performance. Developers and enterprise system owners can utilize various benchmarking tools, each with distinct features and capabilities, to measure these metrics effectively.

GenAI-Perf is a tool for measuring and analyzing the performance indicators, ensuring their LLM deployments are optimized for efficiency and scalability.

How LLM Inference Works

Before examining benchmark metrics, it is crucial to understand the mechanics of LLM inference and the associated terminologies. An LLM application generates results through a multi-stage inference process. Initially, the user submits a query (prompt), which enters a queue, awaiting its turn for processing—this is the queuing phase. Next, the LLM model processes the prompt during the prefill phase. Finally, the model generates a response token by token in the generation phase.

A token, a fundamental concept in LLMs, serves as a core performance metric for LLM inference. It represents the smallest unit of natural language that the LLM processes. The aggregation of all tokens forms the vocabulary, with each LLM employing a tokenizer learned from data to efficiently represent input text. Typically, for many popular LLMs, one token approximates 0.75 English words.

Sequence length pertains to the length of the data sequence. The Input Sequence Length (ISL) denotes the number of tokens the LLM receives, encompassing the user query, system prompts (e.g., model instructions), previous chat history, chain-of-thought reasoning, and documents from the retrieval-augmented generation (RAG) pipeline. The Output Sequence Length (OSL) indicates the number of tokens the LLM generates. Context length refers to the number of tokens the LLM utilizes at each generation step, including both input and output tokens generated thus far. Each LLM has a maximum context length that can be allocated to both input and output tokens.

Streaming is a feature that allows partial LLM outputs to be returned to users incrementally, in chunks of tokens generated so far. This is particularly advantageous for chatbot applications, where receiving an initial response quickly is desirable. While the user processes the partial content, the subsequent chunk of the result is generated in the background. Conversely, in non-streaming mode, the complete answer is returned in a single response.

Metrics

There can be variations in the benchmarking results between different tools. Figure 14 illustrates some of the widely used LLM inference metrics.

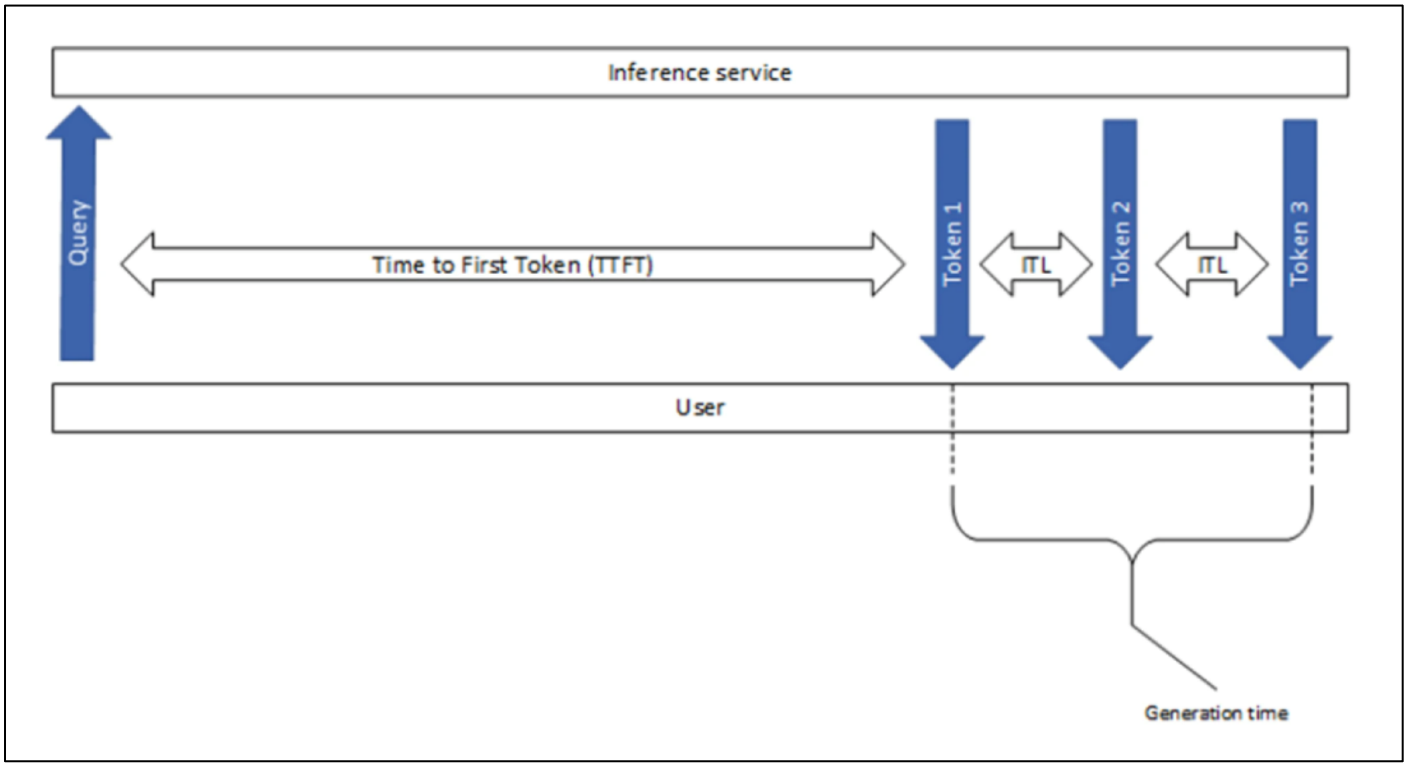
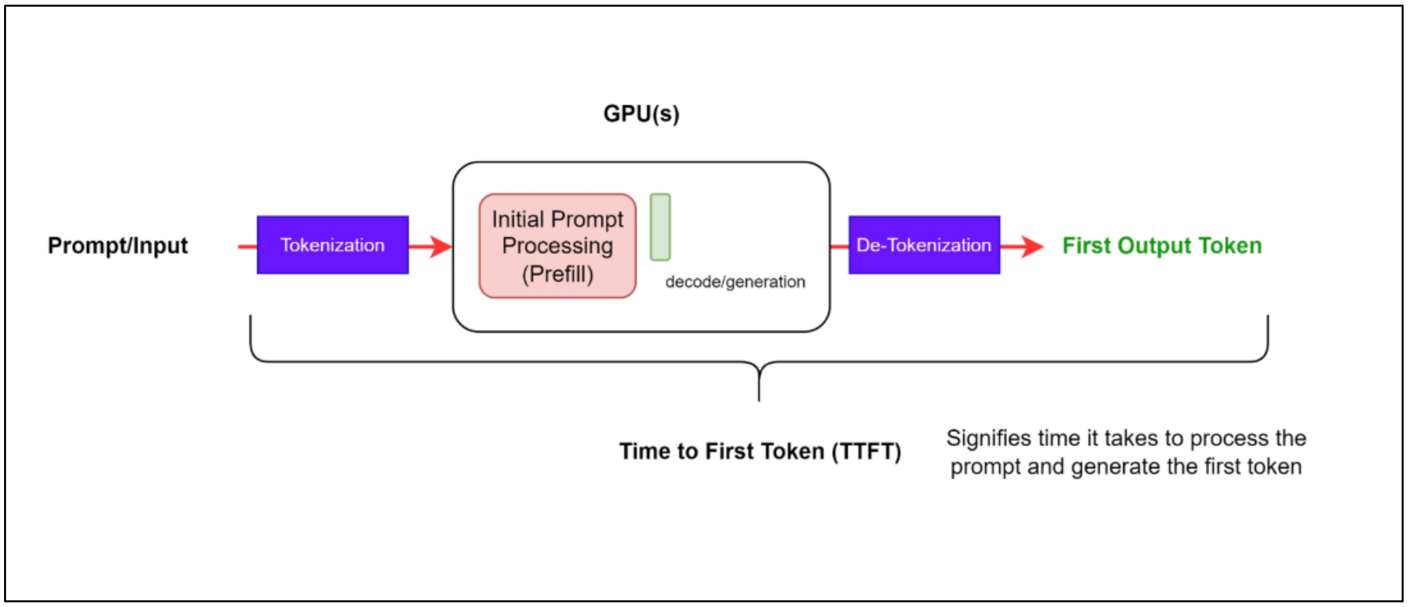


Figure 14. Overview of popular LLM inference performance metrics.

Time to First Token

Time to First Token (TTFT) measures the duration from submitting a query to receiving the first token of the model's output. This metric encompasses request queuing time, prefill time, and network latency. Generally, a longer prompt results in a higher TTFT due to the attention mechanism, which requires the entire input sequence to compute and create the key-value cache (KV-cache) before the iterative generation loop can commence. In production environments, multiple requests may be processed simultaneously, causing the prefill phase of one request to overlap with the generation phase of another, further impacting TTFT.

Figure 15. Overview of popular LLM inference performance metrics.


GPU(s)
Prompt/Input
Tokenization
Initial Prompt Processing (Prefill)
decode/generation
De-Tokenization
First Output Token
Time to First Token (TTFT)
Signifies time it takes to process the prompt and generate the first token

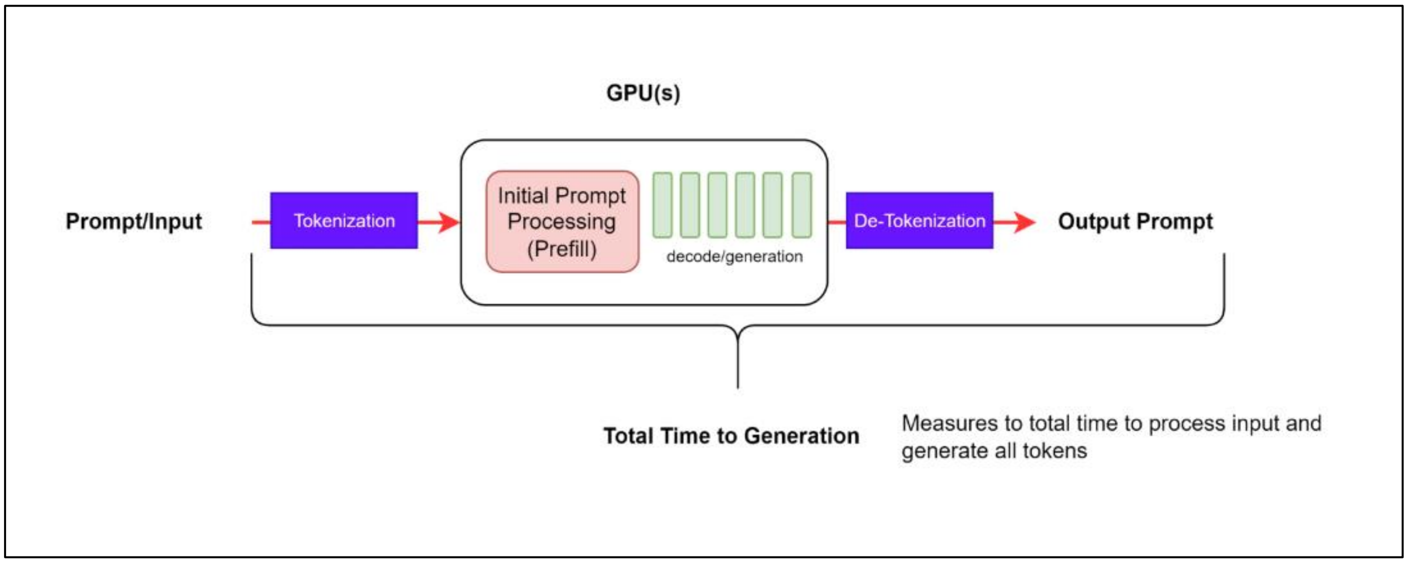
Generation Time

Generation time is the duration from the first token received to the final token received. GenAI-Perf removes the last [done] signal or empty response, so they don't get included in the e2e latency.

End-to-End Request Latency (e2e_latency)

End-to-End Request Latency (e2e_latency) quantifies the total time from submitting a query to receiving the complete response. This metric encompasses the performance of queuing/batching mechanisms and network latencies as demonstrated in Figure 16.

Figure 16. End-to-end Request latency



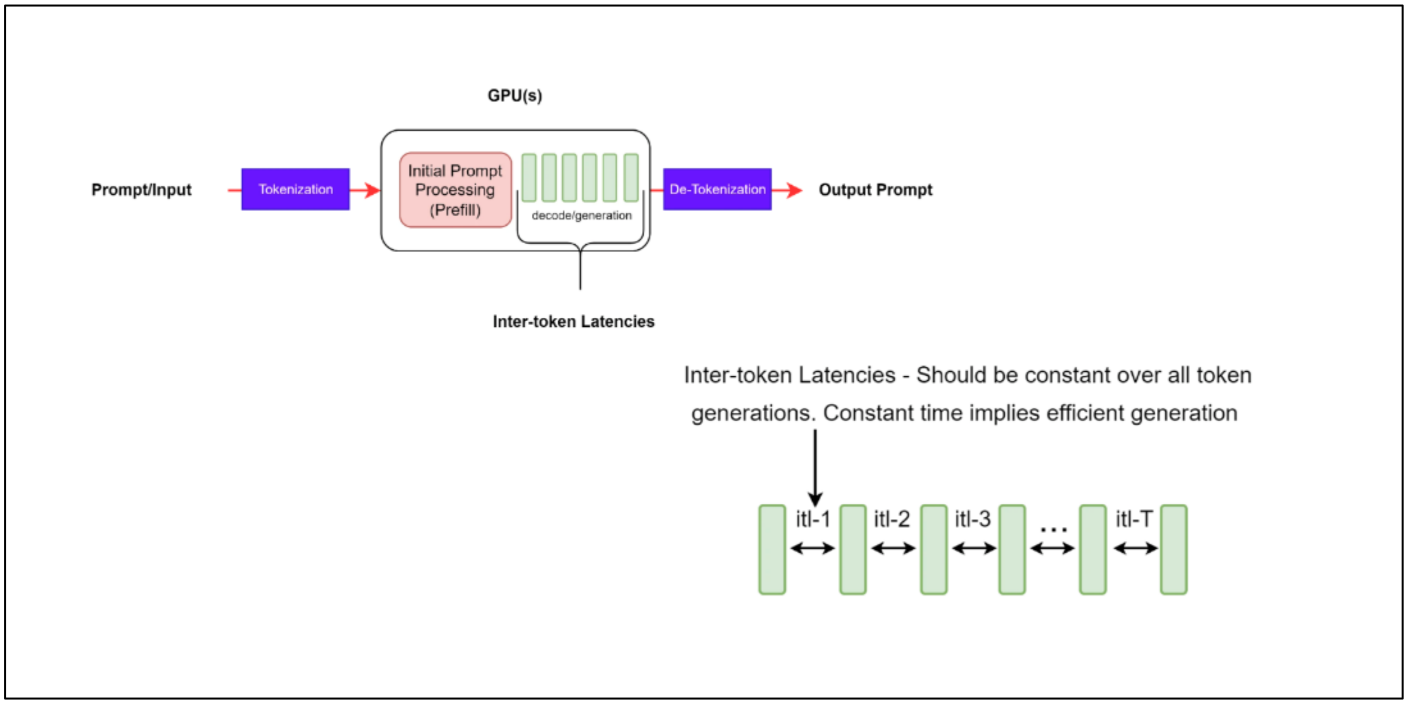
Mathematically, it is expressed as: e2e_latency=TTFT + Generation Time

For an individual request, the end-to-end request latency is the time difference between when the request is sent and when the final token is received. This metric provides a comprehensive measure of the system's responsiveness, accounting for all stages of the request processing pipeline.

Inter Token Latency (ITL)

This is defined as the average time between consecutive tokens and is also known as time per output token (TPOT).

Figure 17. ITL - Latency between successive token generations.


GPU(s)
Prompt/Input
Tokenization
Initial Prompt Processing (Prefill)
decode/generation
De-Tokenization
Output Prompt
Inter-token Latencies
Inter-token Latencies - Should be constant over all token generations. Constant time implies efficient generation
itl-1
itl-2
itl-3
...
itl-T

GenAI-Perf defines ITL as follows:

$$\frac{e2e\_latency - TTFT}{Total\_output\_tokens - 1}$$

The equation for this metric excludes the first token (hence subtracting 1 in the denominator) to ensure that Inter-Token Latency (ITL) reflects only the decoding phase of request processing.

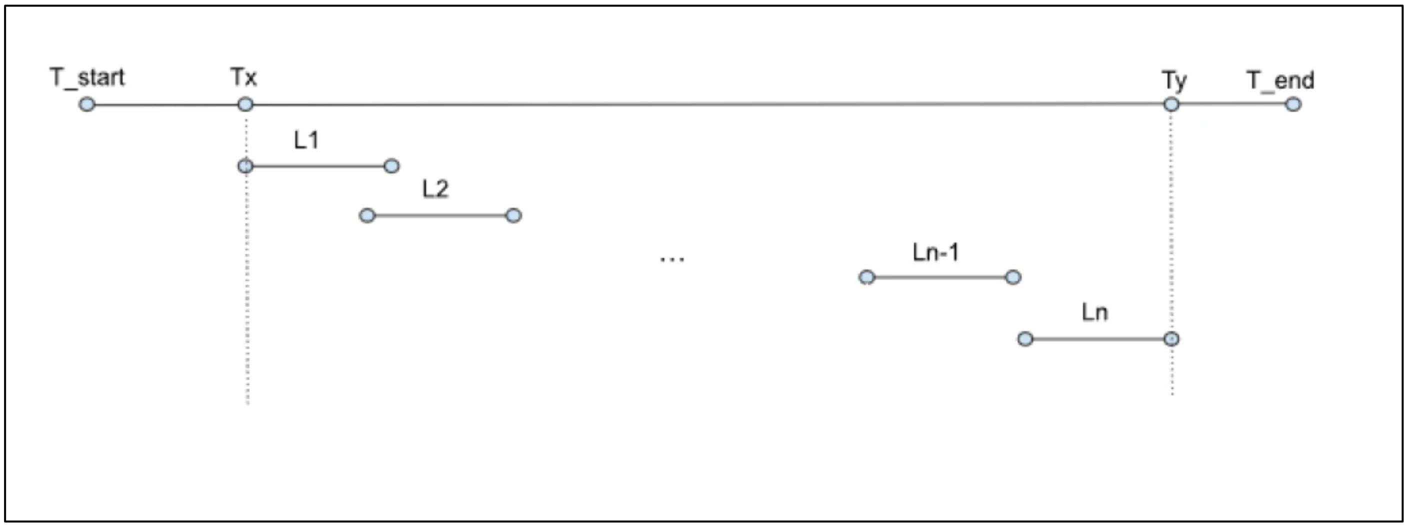
Longer output sequences increase the size of the KV cache, thereby raising memory costs. Additionally, the cost of attention computation grows linearly with the length of the combined input and output sequence for each new token, although this computation is typically not compute-bound. Consistent inter-token latencies indicate efficient memory management, optimal memory bandwidth utilization, and effective attention computation.

Tokens Per Second (TPS)

Total TPS per system represents the total output tokens per seconds throughput, accounting for all the requests happening simultaneously. As the number of requests increases, the total TPS per system increases, until it reaches a saturation point for all the available GPU compute resources, beyond which it might decrease.

Given the following timeline of the entire benchmark with n total requests.

Figure 18. Timeline of events in a benchmarking run



Li : End-to-end latency of i-th request

T_start : start of benchmark

Tx : timestamp of the first request

Ty : timestamp of the last response of the last request

T_end : end of benchmark

GenAI-perf defines the TPS as total output tokens divided by the end-to-end latency between the first request and the last response of the last request.

$$\frac{Total\_output\_tokens}{Ty - Tx}$$

Requests Per Second (RPS)

This is the average number of requests that can be successfully completed by the system in a 1-second period. It is calculated as:

$$RPS = \frac{total\_completed\_requests}{Ty - Tx}$$

Use Cases

An application's specific use cases influence sequence lengths - Input Sequence Length(ISL) and Output Sequence Length(OSL), which in turn affect how efficiently a system processes input to form the KV-cache and generate output tokens. Longer ISL increases memory requirements for the prefill stage, thereby extending TTFT, while longer OSL increases memory requirements (both bandwidth and capacity) for the generation stage, thus raising ITL. Understanding the distribution of inputs and outputs in an LLM deployment is crucial for optimizing hardware utilization. Common use cases and their typical ISL/OSL pairs include:

- Translation

  Involves translating between languages and code, characterized by similar ISL and OSL, typically ranging from 500 to 2000 tokens each.

- Generation

  Encompasses generating code, stories, emails, and generic content via search, characterized by an OSL of O(1000) tokens, significantly longer than an ISL of O(100) tokens.

- Summarization

  Includes retrieval, chain-of-thought prompting, and multi-turn conversations, characterized by an ISL of O(1000) tokens, much longer than an OSL of O(100) tokens.

Real data can also be used as inputs. GenAI-Perf supports datasets such as HuggingFace OpenOrca and CNN Dailymail.

Load Control

- Concurrency (N)

  Refers to the number of concurrent users, each with one active request, or equivalently, the number of requests being served simultaneously by an LLM service. As soon as a user's request receives a complete response, another request is sent, ensuring the system always has exactly N active requests.

  LLMperf sends requests in batches of N but includes a draining period where it waits for all requests to complete before sending the next batch. Consequently, the number of concurrent requests gradually reduces to zero towards the end of the batch. This differs from GenAI-perf, which maintains N active requests throughout the benchmarking period.

  Concurrency is primarily used to describe and control the load on the inference system.

- Max Batch Size

  A batch is a group of simultaneous requests processed by the inference engine, which may be a subset of the concurrent requests. The maximum batch size parameter defines the maximum number of requests the inference engine can process simultaneously.

  If concurrency exceeds the maximum batch size multiplied by the number of active replicas, some requests will queue for later processing, potentially increasing the Time-to-First-Token (TTFT) due to the queuing effect.

- Request Rate

  This parameter controls load by determining the rate at which new requests are sent. A constant (static) request rate (r) means one request is sent every (\frac{1}{r}) seconds, while a Poisson (exponential) request rate determines the average inter-arrival time.

  GenAI-perf supports both concurrency and request rate, but concurrency is recommended. With request rate, the number of outstanding requests may grow unbounded if the requests per second exceed system throughput.

  When specifying concurrencies to test, it is useful to sweep over a range of values, from a minimum of 1 to a maximum not much greater than the max batch size. When concurrency exceeds the max batch size of the engine, some requests will queue, causing system throughput to saturate around the max batch size while latency steadily increases.

Procedure 1. Set up GenAI-Perf

Step 1. Make sure that the NIM LLama-3 inference service is running:

```
[admin@ocp-installer ~]$ oc -n nim-llm get pods
NAME               READY   STATUS    RESTARTS   AGE
pkoppa-nim-llm-0   1/1     Running   0          19h
```

```
[admin@ocp-installer ~]$ oc -n nim-llm get svc
NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
pkoppa-nim-llm         NodePort    172.30.183.181   <none>        8000:31641/TCP   3d8h
pkoppa-nim-llm-sts     ClusterIP   None             <none>        8000/TCP         3d8h.
```

Step 2. Observe that the POD is running. The cluster IP of the service is 172.30.183.181 and it's running on Port 8000. It is also exposed as NodePort on Port 31641.

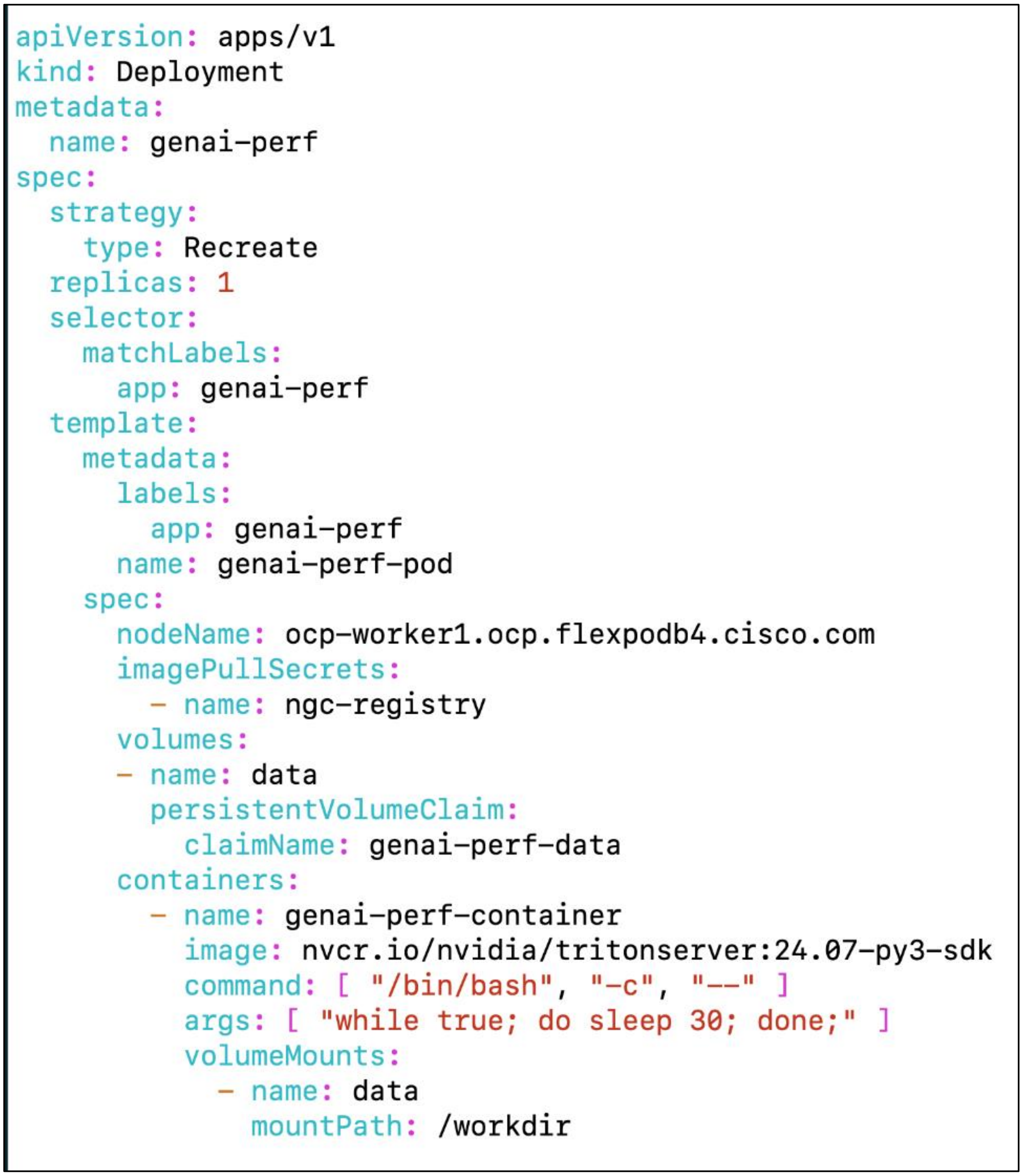
Step 3. Make a test chat completion to make sure that the LLM endpoint is working:

```
curl -X GET 'http://172.30.183.181:8000/v1/models'
curl -X 'POST' \
    'http://172.30.183.181:8000/v1/completions' \
    -H 'accept: application/json' \
    -H 'Content-Type: application/json' \
    -d '{
"model": "meta/llama3-8b-instruct",
"prompt": "Capital of USA",
"max_tokens": 64
}'
```

Once the NIM LLama-3 inference service is running, you can set up a benchmarking tool. The easiest way to install GenAI-Perf is by creating a deployment object in OpenShift using the Triton Server SDK container image.

Step 4. A sample deployment YAML manifest is provided below:

Figure 19. Sample Deployment Manifest



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: genai-perf
spec:
  strategy:
    type: Recreate
  replicas: 1
  selector:
    matchLabels:
      app: genai-perf
  template:
    metadata:
      labels:
        app: genai-perf
      name: genai-perf-pod
    spec:
      nodeName: ocp-worker1.ocp.flexpodb4.cisco.com
      imagePullSecrets:
        - name: ngc-registry
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: genai-perf-data
      containers:
        - name: genai-perf-container
          image: nvcr.io/nvidia/tritonserver:24.07-py3-sdk
          command: [ "/bin/bash", "-c", "--" ]
          args: [ "while true; do sleep 30; done;" ]
          volumeMounts:
            - name: data
              mountPath: /workdir
```

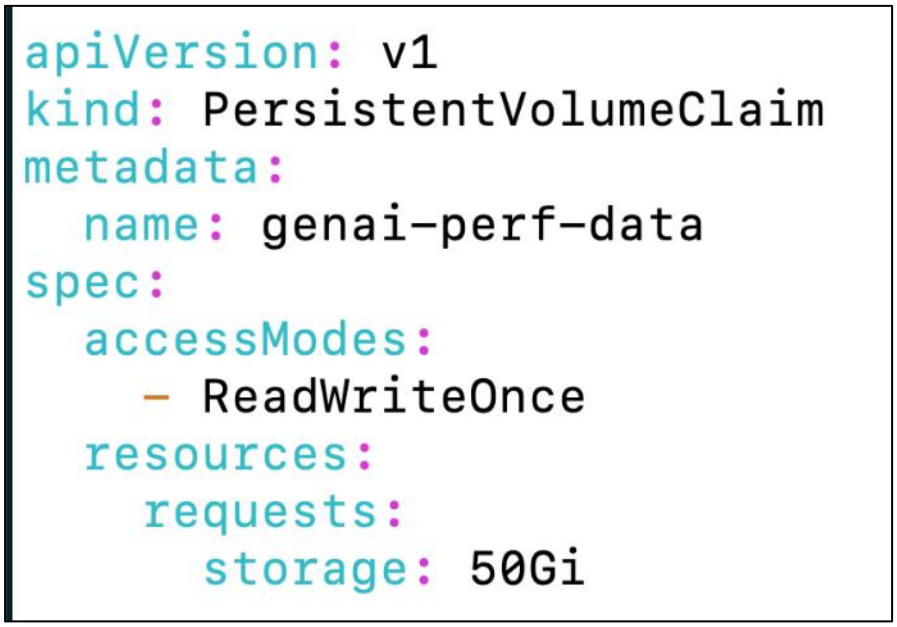
It is recommended to starting a GenAI-perf pod on the same server as NIM to avoid network latency unless you specifically want to factor in the network latency as part of the measurement. Therefore `nodeName: ocp-worker1.ocp.flexpodb4.cisco.com` is passed in pod spec.

Step 5. The image pull secret is required with a NGC API key to download the image. It can be created using:

```
oc create secret docker-registry ngc-registry --docker-server=nvcr.io --docker-username=\$oauthtoken
--docker-password=<API Key>
```

Step 6. A repository is required to store the NeMo check points and other temp folders. A sample PVC configuration to provide persistent storage to the pod is provided below:

Figure 20. Sample PVC



```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: genai-perf-data
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
```

Step 7. Apply the YAML manifests against the OpenShift cluster and make sure all the resources are created:

```
[admin@ocp-installer ~]$ oc -n genaiperf get secrets
NAME           TYPE                             DATA   AGE
ngc-registry   kubernetes.io/dockerconfigjson   1      4d12h
```

```
[admin@ocp-installer ~]$ oc -n genaiperf get pvc
NAME              STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
VOLUMEATTRIBUTESCLASS   AGE
genai-perf-data   Bound    pvc-6be6a2a5-c6d0-4d03-b36e-6906dd968d26   50Gi       RWO            ontap-nfs
<unset>                 4d12h
```

```
[admin@ocp-installer ~]$ oc -n genaiperf get pods
```

```
NAME                            READY   STATUS    RESTARTS   AGE
genai-perf-5ffd6d466b-zrg4n     1/1     Running   0          19h
```

Step 8. From the above output, you can see that the pod is running. Run the following commands to get access to the pod:

```
[admin@ocp-installer ~]$ oc -n genaiperf exec -it genai-perf-5ffd6d466b-zrg4n -- bash
root@genai-perf-5ffd6d466b-zrg4n:/workspace#
```

Step 9. Once inside the pod, you can start the GenAI-perf evaluation harness as follows, which runs a warming load test on the NIM backend:

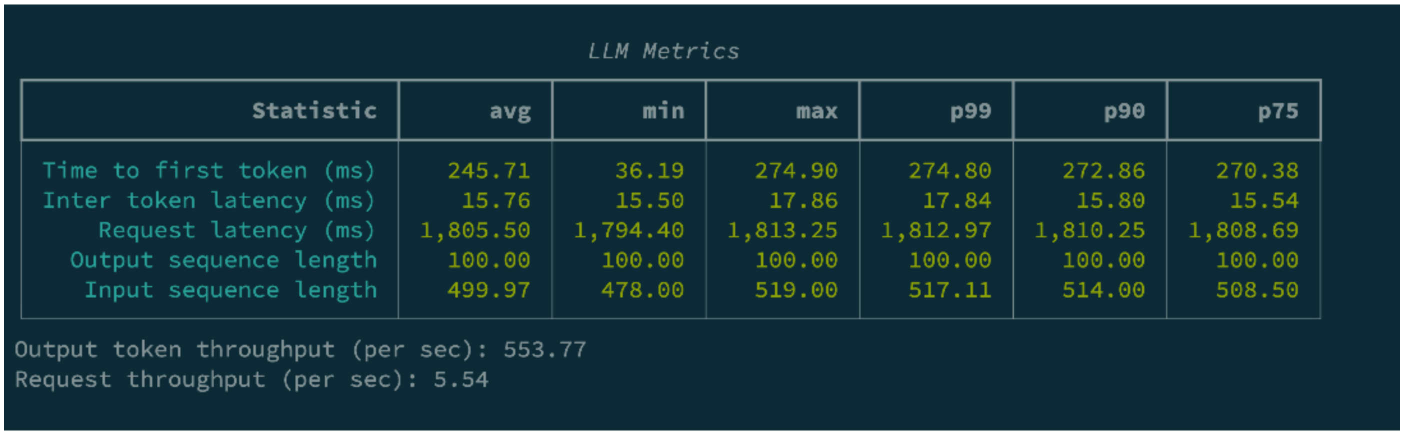
```
export INPUT_SEQUENCE_LENGTH=500
export INPUT_SEQUENCE_STD=10
export OUTPUT_SEQUENCE_LENGTH=100
export CONCURRENCY=10
export MODEL=meta/llama3-8b-instruct


genai-perf \
    -m $MODEL \
    --endpoint-type chat \
    --service-kind openai \
    --streaming \
    -u 172.30.183.181:8000 \
    --synthetic-input-tokens-mean $INPUT_SEQUENCE_LENGTH \
    --synthetic-input-tokens-stddev $INPUT_SEQUENCE_STD \
    --concurrency $CONCURRENCY \
    --output-tokens-mean $OUTPUT_SEQUENCE_LENGTH \
    --extra-inputs max_tokens:$OUTPUT_SEQUENCE_LENGTH \
    --extra-inputs min_tokens:$OUTPUT_SEQUENCE_LENGTH \
    --extra-inputs ignore_eos:true \
    --tokenizer meta-llama/Meta-Llama-3-8B-Instruct \
    -- \
    -v \
    --max-threads=256
```

This example specifies the input and output sequence length and a concurrency to test.

Step 10. This test will use the llama-3 tokenizer from HuggingFace, which is a guarded repository. You will need to apply for access, then login with your HF credential:

```
pip install huggingface_hub
huggingface-cli login
```

Step 11. With a successful execution, you will see similar results in the terminal as shown below:



```
                                        LLM Metrics
┃                Statistic ┃      avg ┃      min ┃      max ┃      p99 ┃      p90 ┃      p75 ┃
│ Time to first token (ms) │   245.71 │    36.19 │   274.90 │   274.80 │   272.86 │   270.38 │
│ Inter token latency (ms) │    15.76 │    15.50 │    17.86 │    17.84 │    15.80 │    15.54 │
│     Request latency (ms) │ 1,805.50 │ 1,794.40 │ 1,813.25 │ 1,812.97 │ 1,810.25 │ 1,808.69 │
│   Output sequence length │   100.00 │   100.00 │   100.00 │   100.00 │   100.00 │   100.00 │
│    Input sequence length │   499.97 │   478.00 │   519.00 │   517.11 │   514.00 │   508.50 │
Output token throughput (per sec): 553.77
Request throughput (per sec): 5.54
```

When the tests complete, GenAI-perf generates the structured outputs in a default directory named "artifacts" under your mounted working directory (/workdir in these examples), organized by model name, concurrency, and input/output length. Your results should look similar to the following:

```
root@genai-perf-5ffd6d466b-zrg4n:/workspace/artifacts# find .
.
./meta_llama3-8b-instruct-openai-chat-concurrency10
./meta_llama3-8b-instruct-openai-chat-concurrency10/llm_inputs.json
./meta_llama3-8b-instruct-openai-chat-concurrency10/profile_export_genai_perf.json
./meta_llama3-8b-instruct-openai-chat-concurrency10/profile_export_genai_perf.csv
./meta_llama3-8b-instruct-openai-chat-concurrency10/profile_export.json
root@genai-perf-5ffd6d466b-zrg4n:/workspace/artifacts#
```

Procedure 2. Sweep through Use Cases

Typically, with benchmarking, a test would be set up to sweep over a number of use cases, such as input/output length combinations, and load scenarios, such as different concurrency values.

Step 1. Use the following bash script to define the parameters so that GenAI-perf executes through all the combinations.

```
declare -A useCases

# Populate the array with use case descriptions and their specified input/output lengths
useCases["Translation"]="200/200"
useCases["Text classification"]="200/5"
useCases["Text summary"]="1000/200"

# Function to execute genAI-perf with the input/output lengths as arguments
runBenchmark() {
    local description="$1"
    local lengths="${useCases[$description]}"
    IFS='/' read -r inputLength outputLength <<< "$lengths"

    echo "Running genAI-perf for$descriptionwith input length$inputLengthand output length$outputLength"
    #Runs
    for concurrency in 1 2 5 10 50 100 250; do

        local INPUT_SEQUENCE_LENGTH=$inputLength
        local INPUT_SEQUENCE_STD=0
        local OUTPUT_SEQUENCE_LENGTH=$outputLength
        local CONCURRENCY=$concurrency
        local MODEL=meta/llama3-8b-instruct

        genai-perf \
            -m $MODEL \
            --endpoint-type chat \
            --service-kind openai \
            --streaming \
            -u 172.30.183.181:8000 \
            --synthetic-input-tokens-mean $INPUT_SEQUENCE_LENGTH \
            --synthetic-input-tokens-stddev $INPUT_SEQUENCE_STD \
            --concurrency $CONCURRENCY \
            --output-tokens-mean $OUTPUT_SEQUENCE_LENGTH \
            --extra-inputs max_tokens:$OUTPUT_SEQUENCE_LENGTH \
            --extra-inputs min_tokens:$OUTPUT_SEQUENCE_LENGTH \
            --extra-inputs ignore_eos:true \
            --tokenizer meta-llama/Meta-Llama-3-8B-Instruct \
            --measurement-interval 10000 \
            --profile-export-file ${INPUT_SEQUENCE_LENGTH}_${OUTPUT_SEQUENCE_LENGTH}.json \
            -- \
            -v \
            --max-threads=256

    done
}

# Iterate over all defined use cases and run the benchmark script for each
for description in "${!useCases[@]}"; do
    runBenchmark "$description"
done
```

Step 2. Save this script in a working directory, such as under /workdir/benchmark.sh. You can then execute it with the following command:

```
cd /workdir
bash benchmark.sh
```

The following tables lists the details of performance for three different use cases.

Table 13. NIM for LLM Benchmarking for Generation Use Case (I/O = 1000/200)

Number of GPUs	Concurrency	Time to First Token (ms)	Inter-token Latency (ms)	End-to-End Request Latency (ms)	Output Token Throughput (Tokens Per Second)	Request Throughput (Requests Per Second)
1 X L40S	1	58.64	14.75	2993.93	66.8	0.33
	2	91.69	14.92	3061.35	130.62	0.65
	5	238.71	15.72	3366.86	296.97	1.48
	10	519.59	16.4	3783.3	528.56	2.64
	25	1430.98	18.15	5043	991.26	4.96
	50	2223.81	24.72	7143.2	1398.09	6.99

Number of GPUs	Concurrency	Time to First Token (ms)	Inter-token Latency (ms)	End-to-End Request Latency (ms)	Output Token Throughput (Tokens Per Second)	Request Throughput (Requests Per Second)
	100	3202.2	42.89	11737.23	1523.29	7.62
2 X L40S	1	80.05	9.24	1917.86	104.27	0.52
	2	118.46	9.55	2018.88	198.08	0.99
	5	306.56	11.05	2505.44	399.08	2
	10	654.43	12.11	3064.37	652.51	3.26
	25	1726.59	13.79	4470.19	1118.26	5.59
	50	2808.82	20.68	6923.89	1441.51	7.21
	100	5496.26	31.46	11757.24	1698.14	8.49

Table 14. NIM for LLM Benchmarking for Summarization Use Case (I/O = 200/5)

Number of GPUs	Concurrency	Time to First Token (ms)	Inter-token Latency (ms)	End-to-End Request Latency (ms)	Output Token Throughput (Tokens Per Second)	Request Throughput (Requests Per Second)
1 X L40S	1	20.09	14.47	77.98	64.07	12.81
	2	34.29	14.99	94.25	105.96	21.19
	5	58.6	17	126.61	197.29	39.46
	10	110.22	17.76	181.25	275.67	55.13
	25	173.33	42.72	344.2	362.89	72.58
	50	434.64	52.66	645.27	387.34	77.47
	100	855.6	104.41	1273.24	391.03	78.21
2 X L40S	1	23.31	8.88	58.83	84.89	16.98
	2	36.68	10.54	78.86	126.67	25.33

Number of GPUs	Concurrency	Time to First Token (ms)	Inter-token Latency (ms)	End-to-End Request Latency (ms)	Output Token Throughput (Tokens Per Second)	Request Throughput (Requests Per Second)
	5	75.11	12.8	126.3	197.79	39.56
	10	143.68	14.06	199.9	249.98	50
	25	328.88	22.79	420.02	297.4	59.48
	50	560.9	57.45	790.7	315.98	63.2
	100	1197.09	87.61	1547.54	322.89	64.58

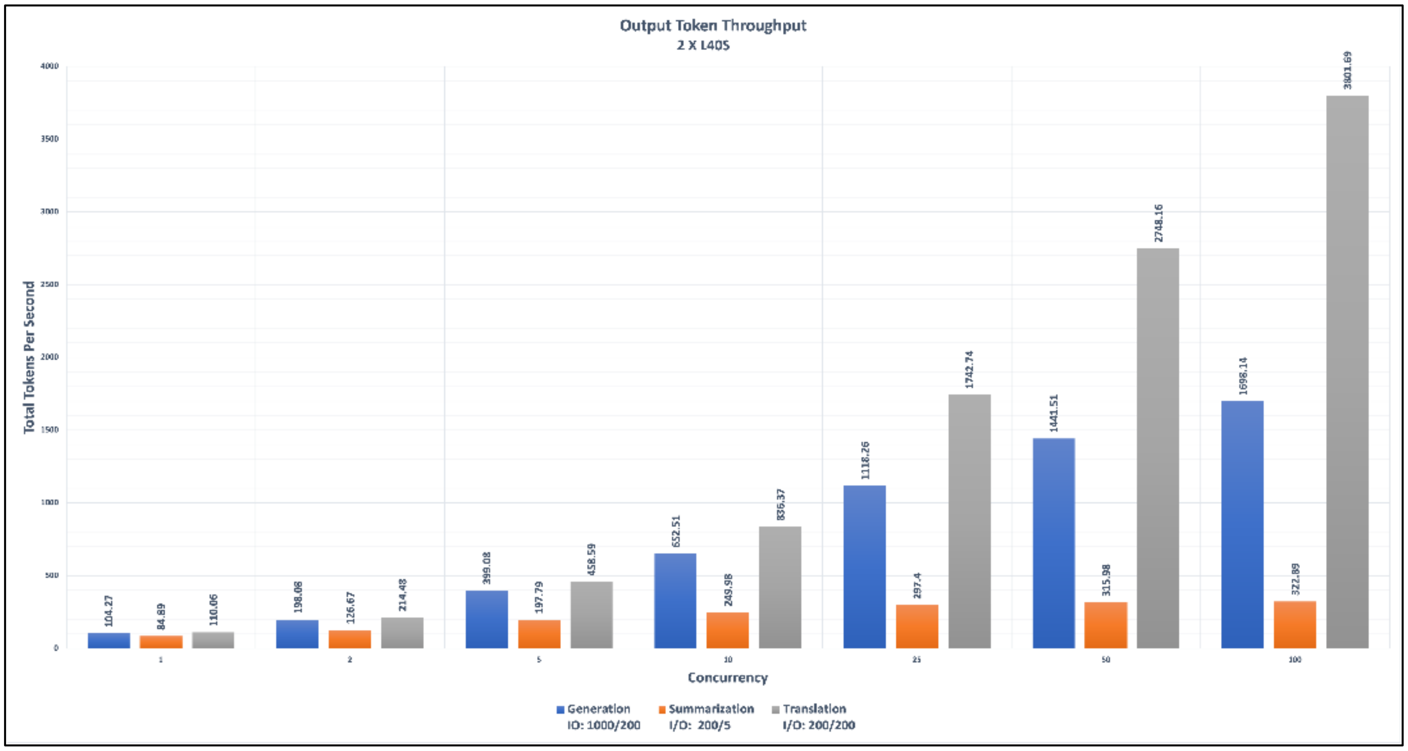
Table 15. NIM for LLM Benchmarking for Translation Use Case (I/O = 200/200)

Number of GPUs	Concurrency	Time to First Token (ms)	Inter-token Latency (ms)	End-to-End Request Latency (ms)	Output Token Throughput (Tokens Per Second)	Request Throughput (Requests Per Second)
1 X L40S	1	21.35	14.51	2908.4	68.76	0.34
	2	33.85	14.56	2931.88	136.39	0.68
	5	57.89	14.98	3038.66	329.04	1.65
	10	109.57	15.22	3137.97	637.23	3.19
	25	229.13	16.25	3463.21	1442.89	7.21
	50	434.81	17.8	3977.64	2512.5	12.56
	100	996.21	20.84	5143.22	3884.17	19.42
2 X L40S	1	24.11	9.01	1816.91	110.06	0.55
	2	36.65	9.18	1864.46	214.48	1.07
	5	75.07	10.58	2180.26	458.59	2.29
	10	143.17	11.3	2390.88	836.37	4.18
	25	289.17	12.95	2867.12	1742.74	8.71

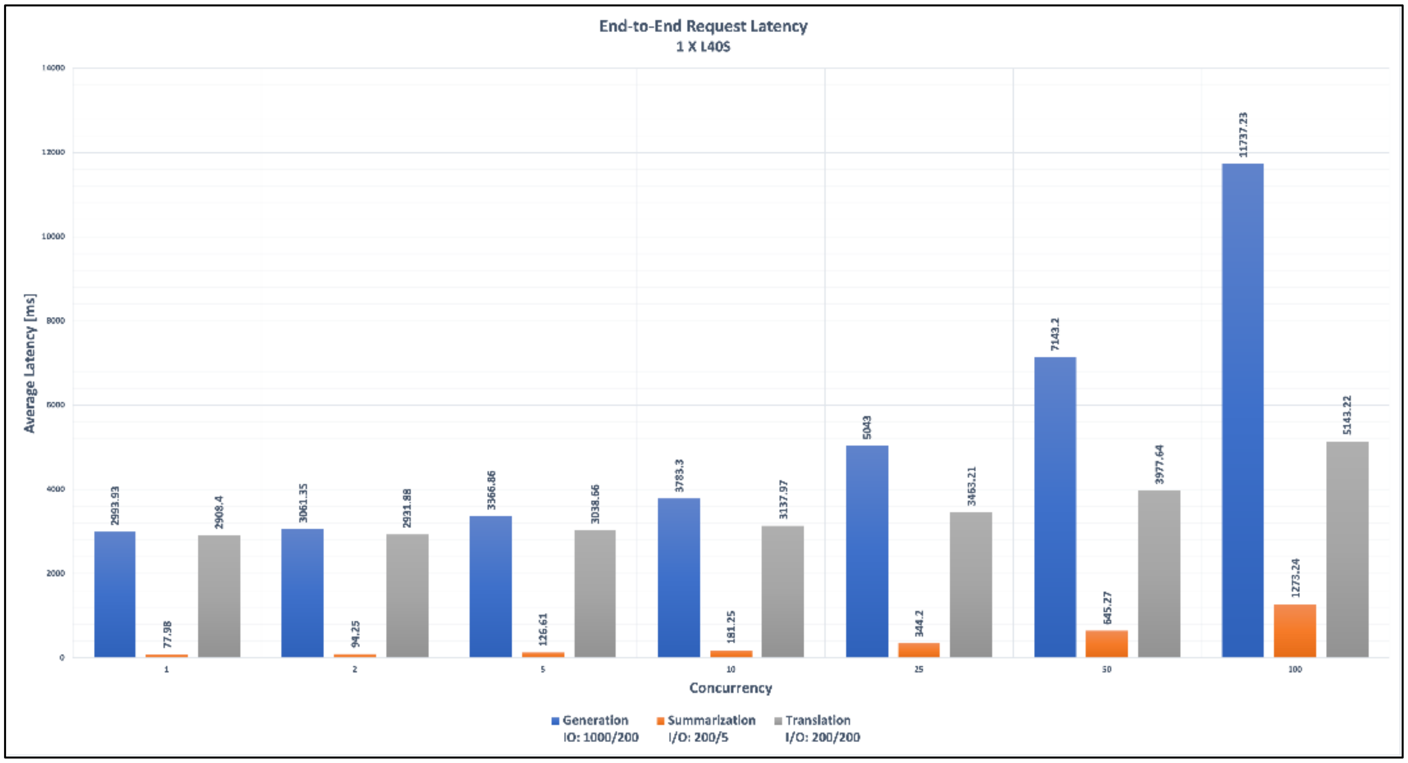
Number of GPUs	Concurrency	Time to First Token (ms)	Inter-token Latency (ms)	End-to-End Request Latency (ms)	Output Token Throughput (Tokens Per Second)	Request Throughput (Requests Per Second)
	50	550.76	15.5	3635.91	2748.16	13.74
	100	1236.57	20.18	5253.18	3801.69	19.01

Analyzing the Output

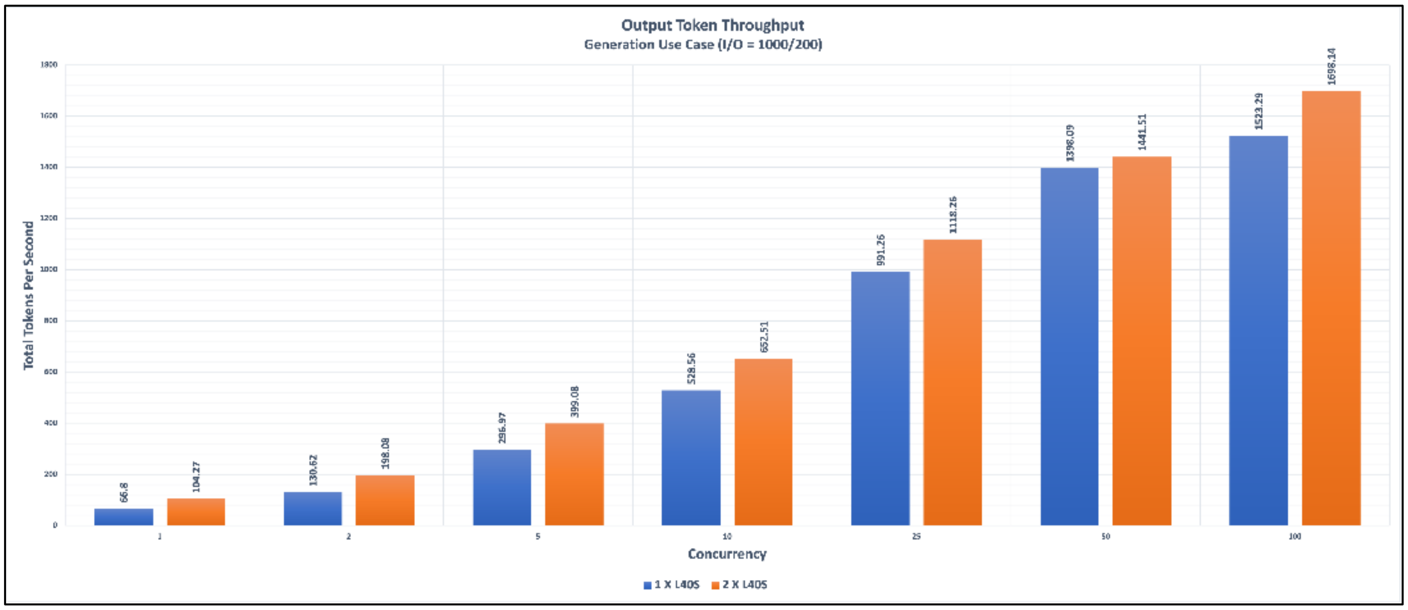
The figure below shows the total output tokens per seconds throughput, accounting for all the requests happening simultaneously from all 3 uses case.



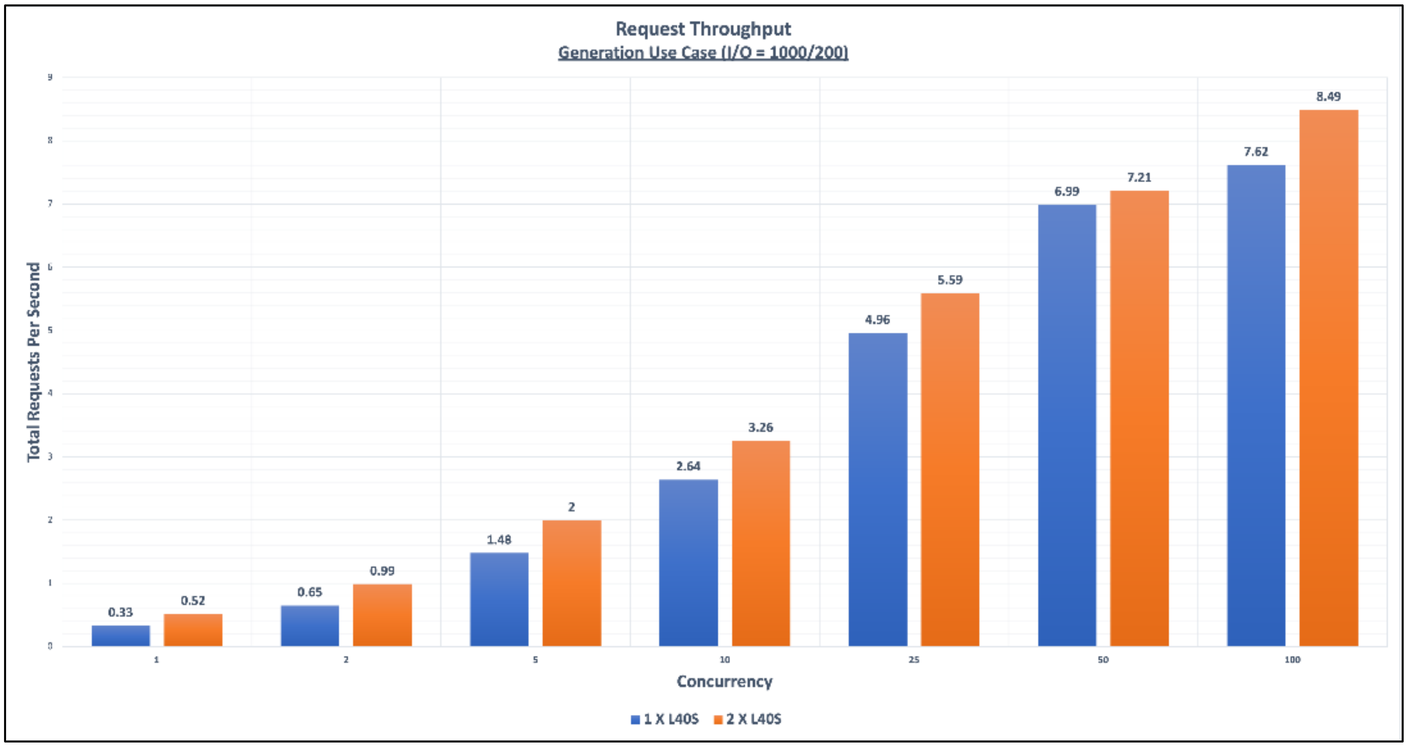
The figure below shows how long it takes from submitting a query to receiving the full response for 3 different use cases.



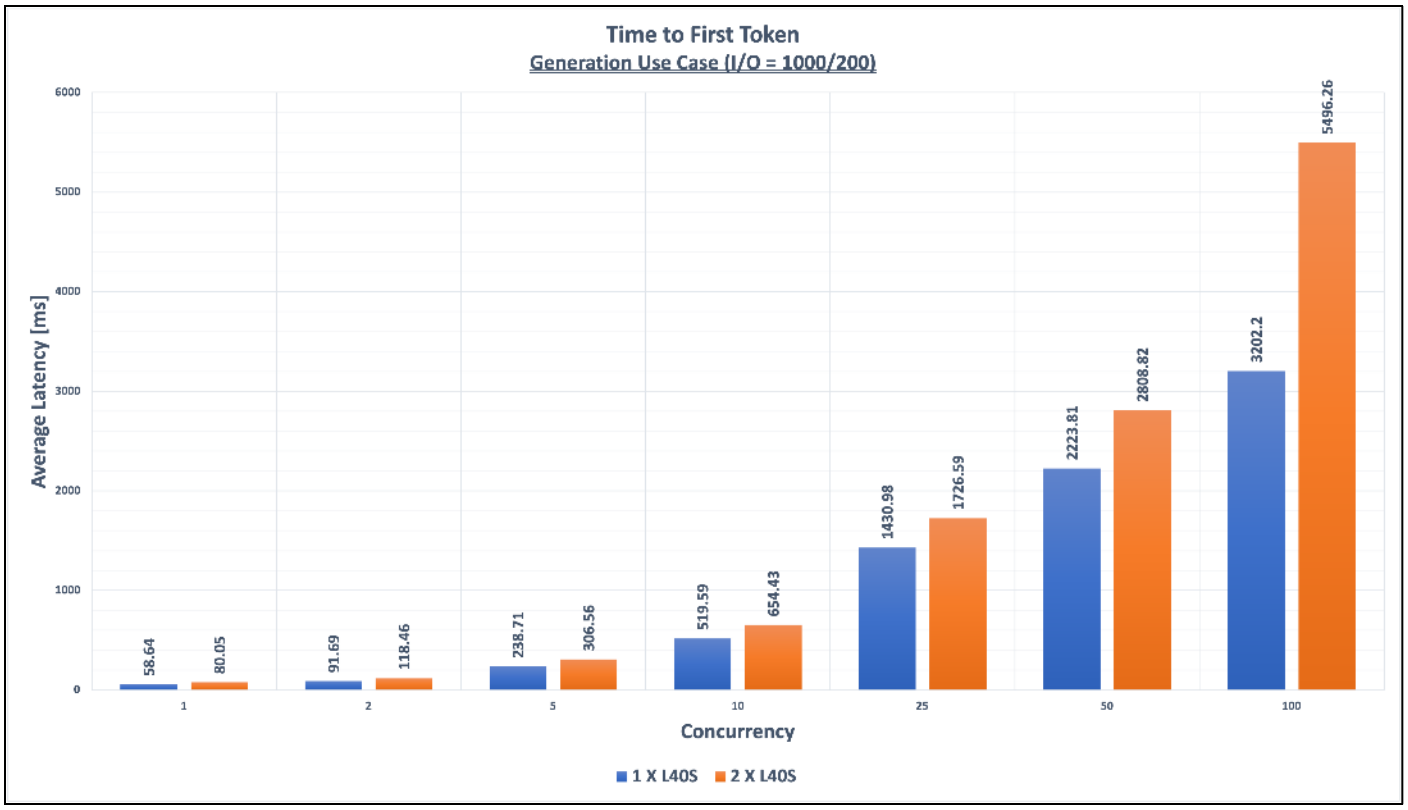
The figure below shows the total output tokens per seconds throughput, accounting for all the requests happening simultaneously with 1 L40S GPU vs 2 X L40S GPU.



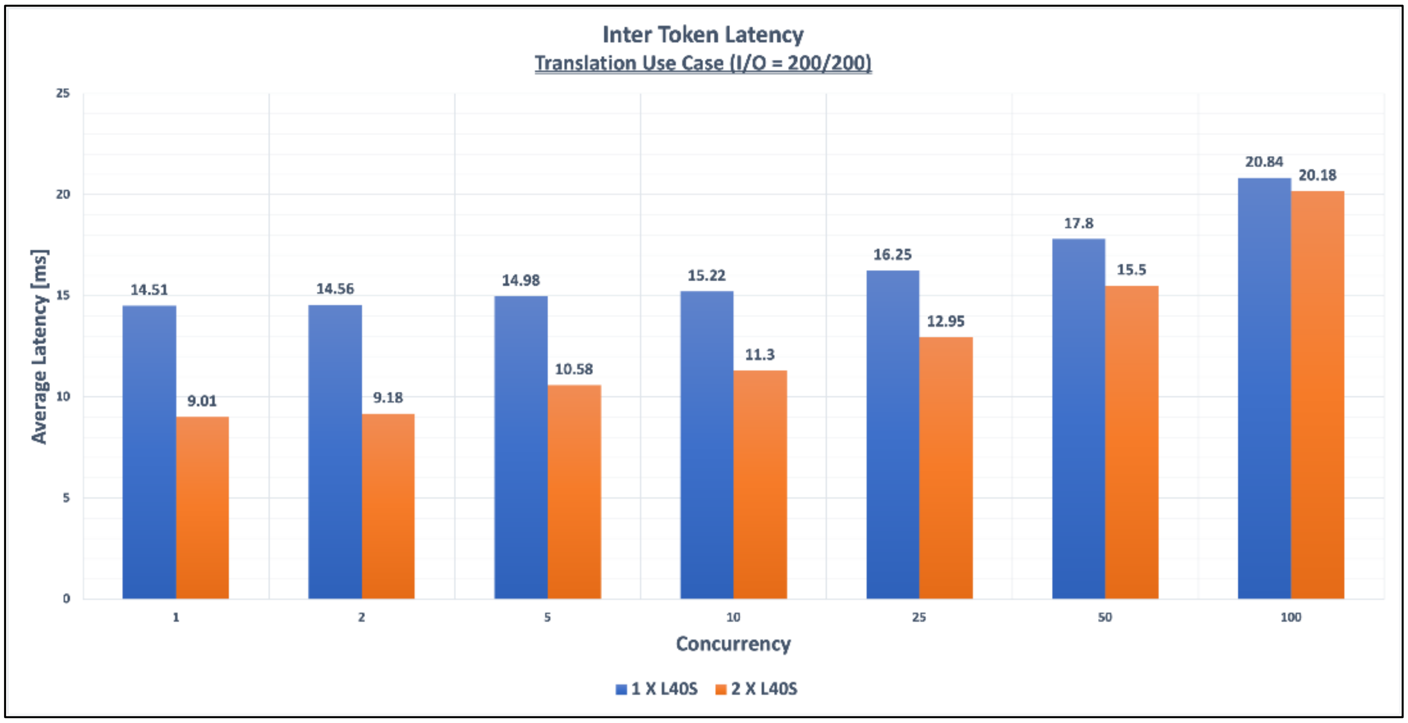
The figure below shows the average number of requests that can be successfully completed by the system in a 1-second period with 1 L40S GPU vs 2 X L40S GPU.



The figure below shows how long a user needs to wait before seeing the model's output with 1 L40S GPU vs 2 X L40S GPU.



The figure below shows the average time between consecutive tokens with 1 L40S GPU vs 2 X L40S GPU.



Interpreting the Results

The following plots illustrate the Latency-Throughput curves for three different use cases with different Input Sequence Length and Output Sequence Lengths.

Generation (I/O : 1000/200), **Summarization** (I/O : 200/5), and **Translation** (I/O : 200/200).

Figure 21. Latency-Throughput Curve for Generation (I/O : 1000/200)

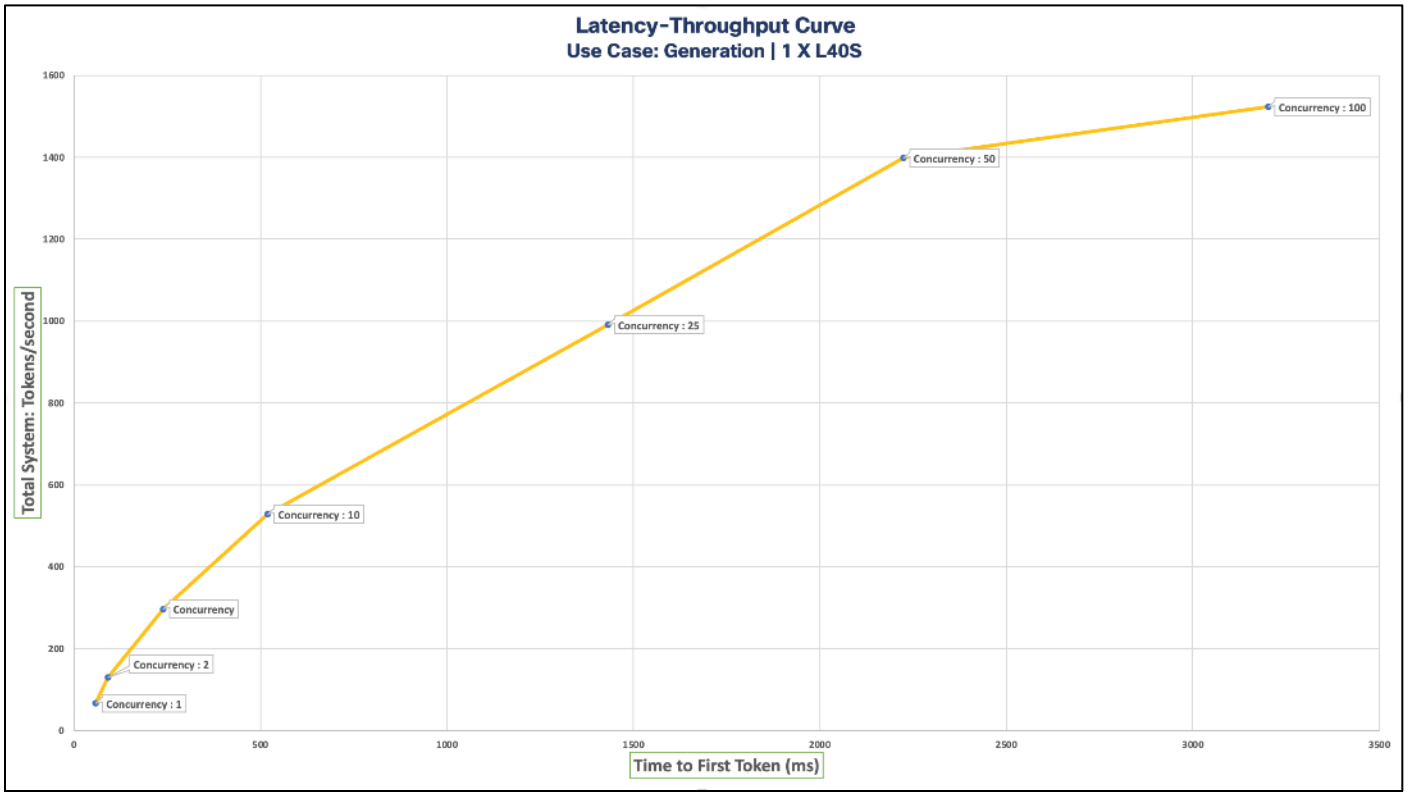


Figure 22. Latency-Throughput Curve for Summarization (I/O : 200/5)

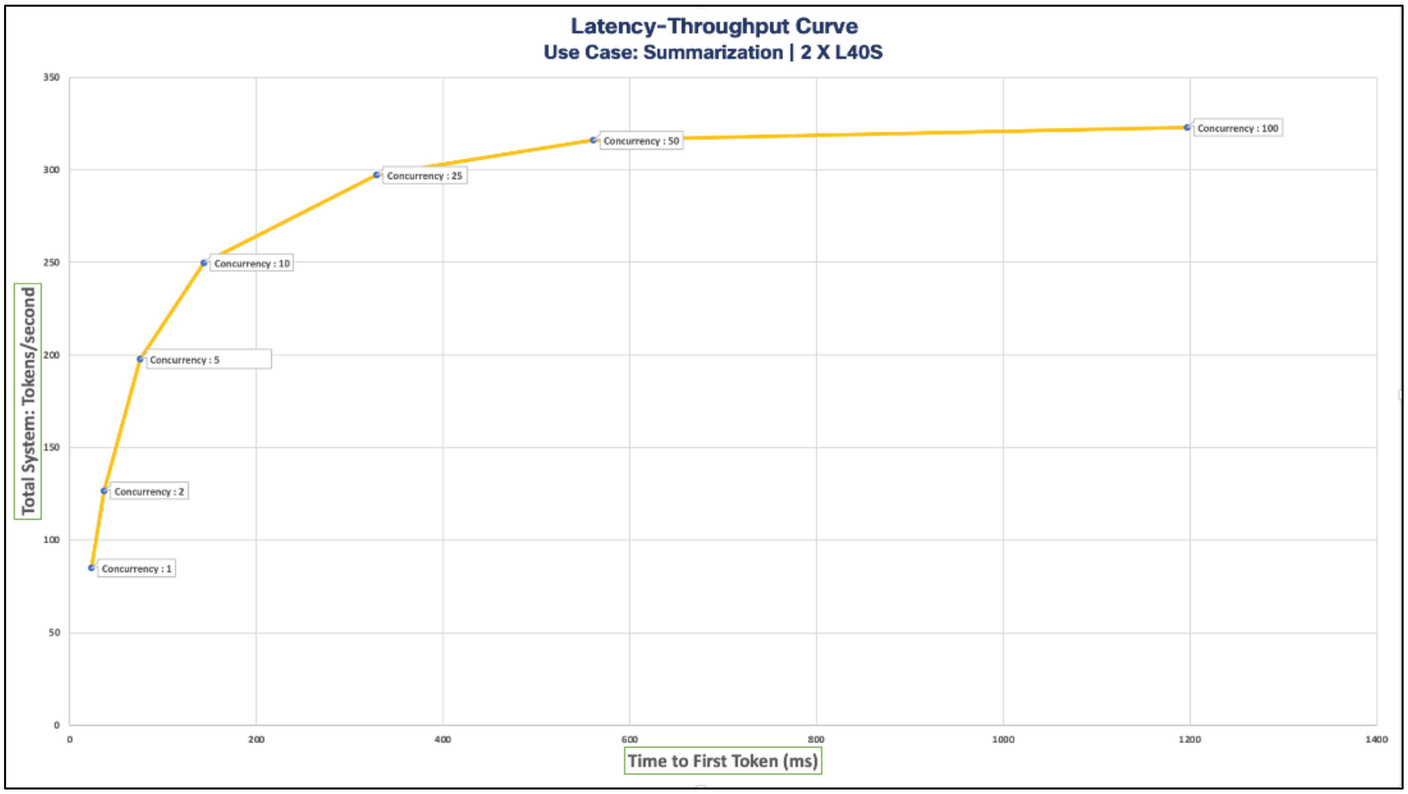
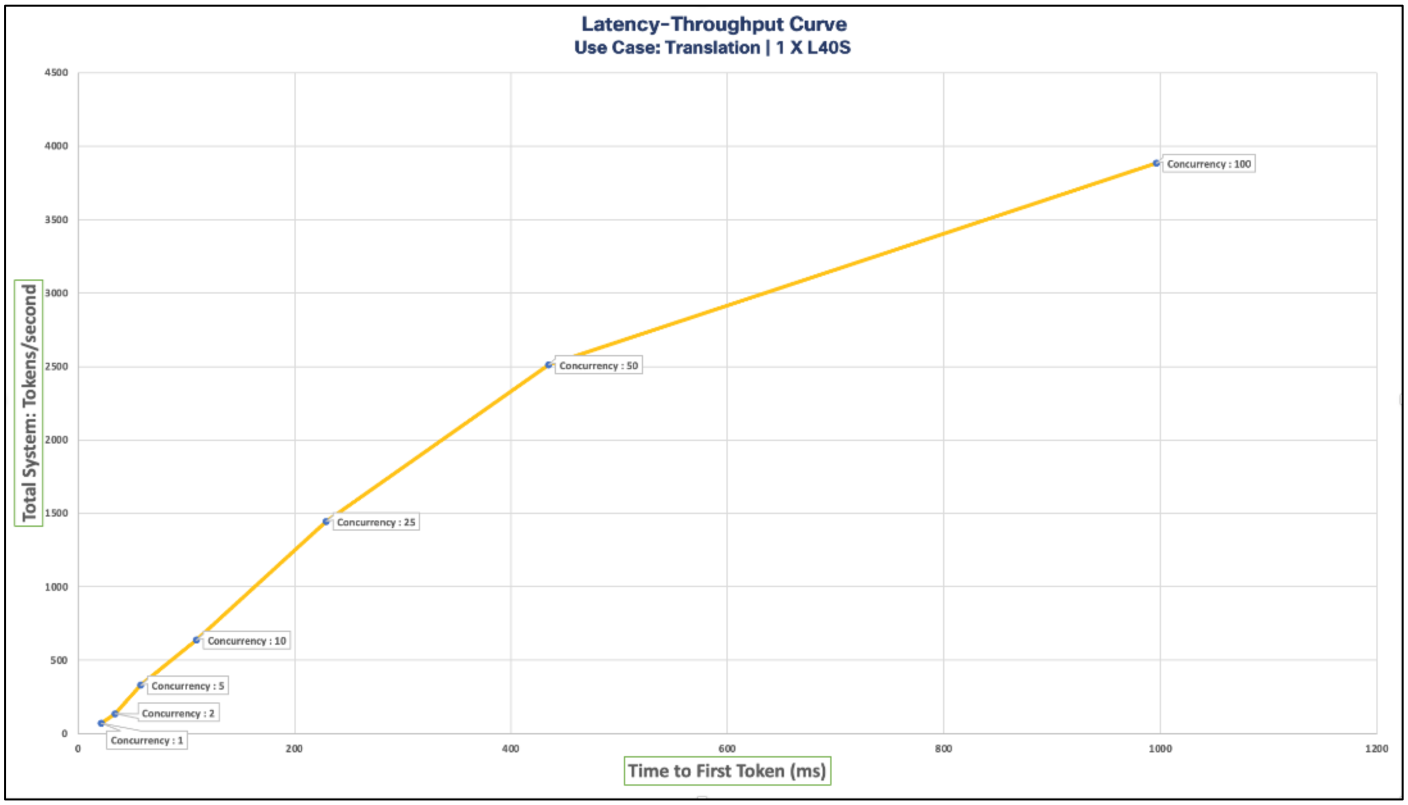


Figure 23. **Latency-Throughput Curve for Translation (I/O : 200/200)**



- Axes Definitions
  - X-axis: Time to First Token (TTFT) - The duration a user waits before the model starts generating output.
  - Y-axis: Total System Throughput - The total number of output tokens generated per second.
  - Dots: Represent different concurrency levels.

Usage Guidelines

- Latency Budget Approach
  - Objective: Determine the highest achievable throughput within a specified latency limit.
  - Method: Identify the maximum acceptable TTFT on the X-axis, then find the corresponding Y value and concurrency level. This indicates the highest throughput achievable within the given latency constraint.
- Concurrency-Based Approach
  - Objective: Understand the latency and throughput for a specific concurrency level.
  - Method: Locate the desired concurrency level on the plot. The intersecting X and Y values represent the latency and throughput for that concurrency.
- Key Observations

  The Latency-Throughput Curve for Summarization plot highlights concurrency levels where latency increases significantly with minimal or no throughput gain. Concurrency = 50 is an example of such a point.

- Alternative Metrics

  Similar plots can be generated using different metrics on the X-axis, such as ITL (Inference Time Latency), e2e_latency (End-to-End Latency), or TPS_per_user (Transactions Per Second per User). These plots help visualize the trade-offs between total system throughput and individual user latency.

Sizing

Memory Calculations for LLM Inferencing

Total Memory required is the sum of model memory size and the KV cache.

Calculations for the required total memory are provided below:

```
Model Memory Size = Model Parameters * Precision

KV Cache Size = 2 x Batch Size x Context Size  x Number of Layers x Model Dimensions x Precision

Total Memory Requirements (GB) = Model Memory Size (GB) + KV Cache Size (GB)
```

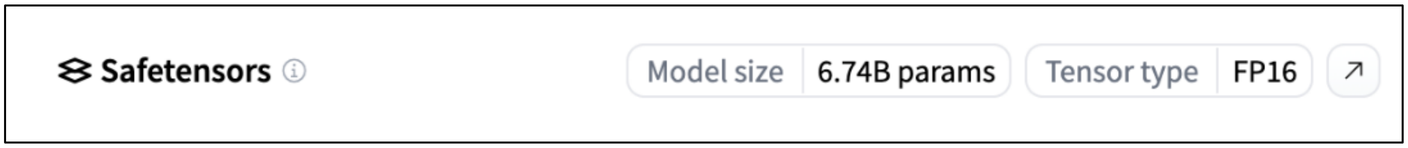
For some models, model dimension data might not be available. In that case, model dimension can be calculated as:

```
Model Dimensions = Attention Head Size X Number of Attention Heads
```

Model Parameters, Precision, Number of layers, Model Dimension are specific to models, and it can be found in the Model card for the model.

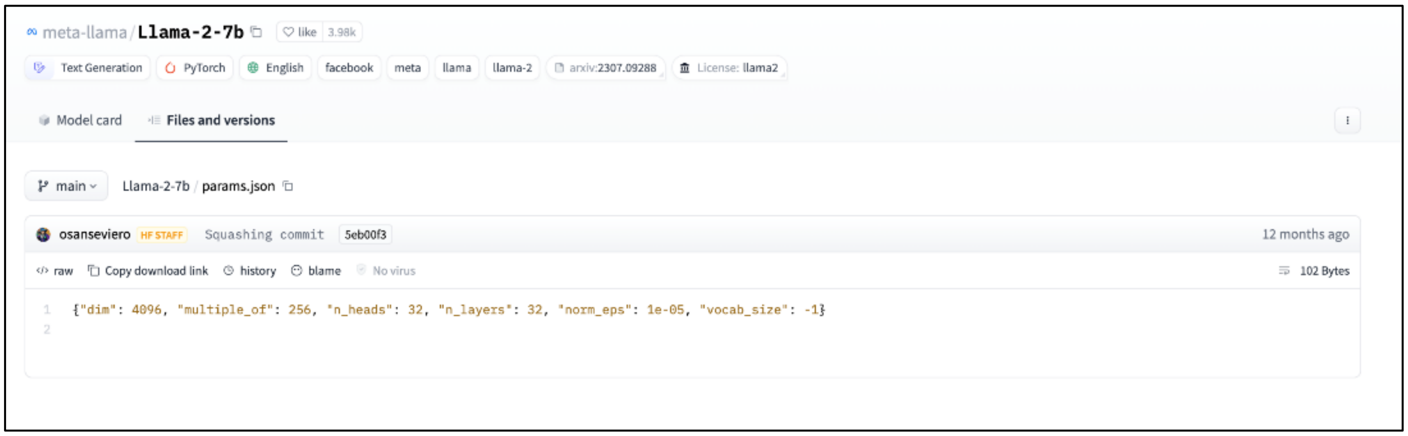
Context Size and batch size are input from users.

The following is an example memory calculation for Llama 2:



Model size 6.74B params

Tensor type FP16



Llama-2-7b / params.json

osanseviero HF STAFF Squashing commit 5eb00f3 12 months ago

102 Bytes

```
{"dim": 4096, "multiple_of": 256, "n_heads": 32, "n_layers": 32, "norm_eps": 1e-05, "vocab_size": -1}
```

For the Llama 2 model:

```
Total model parameters:    6.74B Parameters.

Precision:                 FP16. (2 Bytes)

Number of layers:          32

Model Dimension:           4096
```

Therefore, the model memory is calculated as shown below:

```
Model Memory Size = Model Parameters * Precision



Model Memory Size for Llama 2 = 6,740,000,000 * 2 Bytes/Parameter

                              = 13,480,000,000 Bytes

                              = 13.48 Giga Bytes
```

Also, considering an example of maximum Input Tokens Length of 1024, Maximum Output Tokens Length of 1024, and the Batch size of 8, below are the calculations for KV Cache Size:

```
KV Cache Size  = 2 x Batch Size x Context Size  x Number of Layers x Model Dimensions x Precision

KV Cache Size  = 2 x 8 x (1024+1024) x 32 x 4096 x 2 Bytes/Parameter

              = 8,589,934,592 Bytes

              =  8.59 Giga Bytes
```

Therefore, Llama2 with maximum Input Tokens Length of 1024, Maximum Output Tokens Length of 1024, and the Batch size of 8, the total memory required is shown below:

```
Total Memory Requirements (GB) = Model Memory Size (GB) + KV Cache Size (GB)

                           = 13.48 + 8.59 Giga Bytes

                           = 22.07 Giga Bytes
```

Performance Calculations

Note: The performance benchmark can be run on model.

Based on the performance requirement, number of users, number of input and output tokens, latency and throughput required, you can choose the appropriate Large Language Model, Inferencing backend, GPUs, and compute infrastructure.

Performance of the model depends on the prefill and decode phases. These two phases have different impacts on the performance of the LLM. While the prefill phase effectively saturates GPU compute at small batch sizes, the decode phase results in low compute utilization as it generates one token at a time per request.

The prefill phase is compute-bound, while the decode phase is memory-bound. So, the following factors need to be considered and measured:

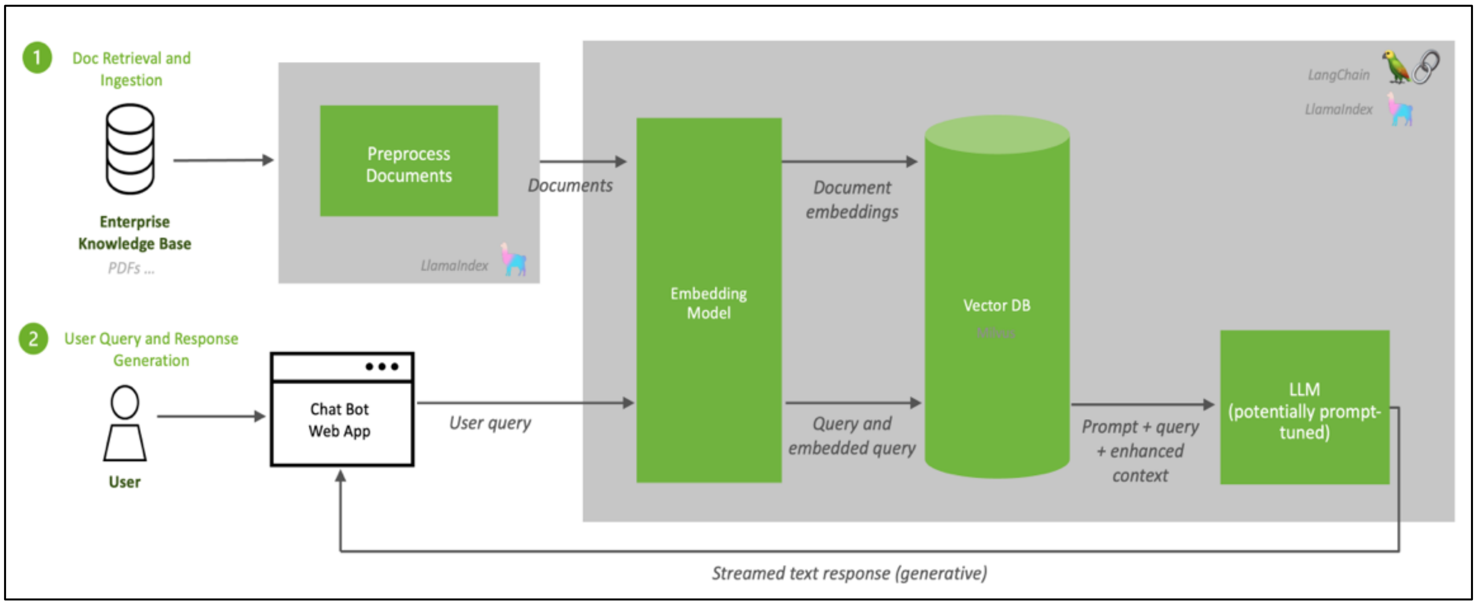
- Prefill Latency
- Prefill Throughput
- Decode Total Latency
- Decode Token Latency
- Decode Throughput

The performance benchmark can be run with different sizes (1,2,4,8,10,25,250, 100 and so on). Also, separate tests can be run focused on performance comparison between 2 different models.

RAG Evaluation

How RAG Works

Retrieval-Augmented Generation (RAG) is a technique used to enrich LLM outputs by using additional relevant information from an external knowledge base. This allows an LLM to generate responses based on context be-yond the scope of its training data.


1
Doc Retrieval and Ingestion
Enterprise Knowledge Base
PDFs ...
Preprocess Documents
LlamaIndex
Documents
LangChain
LlamaIndex
Embedding Model
Document embeddings
Vector DB
2
User Query and Response Generation
User
Chat Bot Web App
User query
Query and embedded query
Prompt + query + enhanced context
LLM (potentially prompt-tuned)
Streamed text response (generative)

Why Evaluate RAG?

RAG enhances content generation by leveraging existing information effectively. It can amalgamate specific, relevant details from multiple sources to generate more accurate and relevant query results. This makes RAG potentially invaluable in various domains, including content creation, question & answer applications, and information synthesis. RAG does this by combining the strengths of retrieval, usually using dense vector search, and text generation, but to see what is and isn't working in your RAG system, to refine and optimize, you must evaluate it.

Note: Evaluation is essential to validate and make sure your application does what is expected.

Evaluation Process

Evaluating RAG goes through the following steps:

1. Choose Right Evaluation Framework.
2. Generate Synthetic Dataset.
3. Generate RAG Outputs for the Questions from Synthetic Dataset.
4. Evaluating Using the Generated Dataset.
5. Understanding The Metrics.

Choose the Right Evaluation Framework

RAG evaluation quantifies the accuracy of retrieval phrase by calculating metrics on the top results your system returns, enabling you to programmatically monitor pipeline's precision, recall ability, and faithfulness to facts.

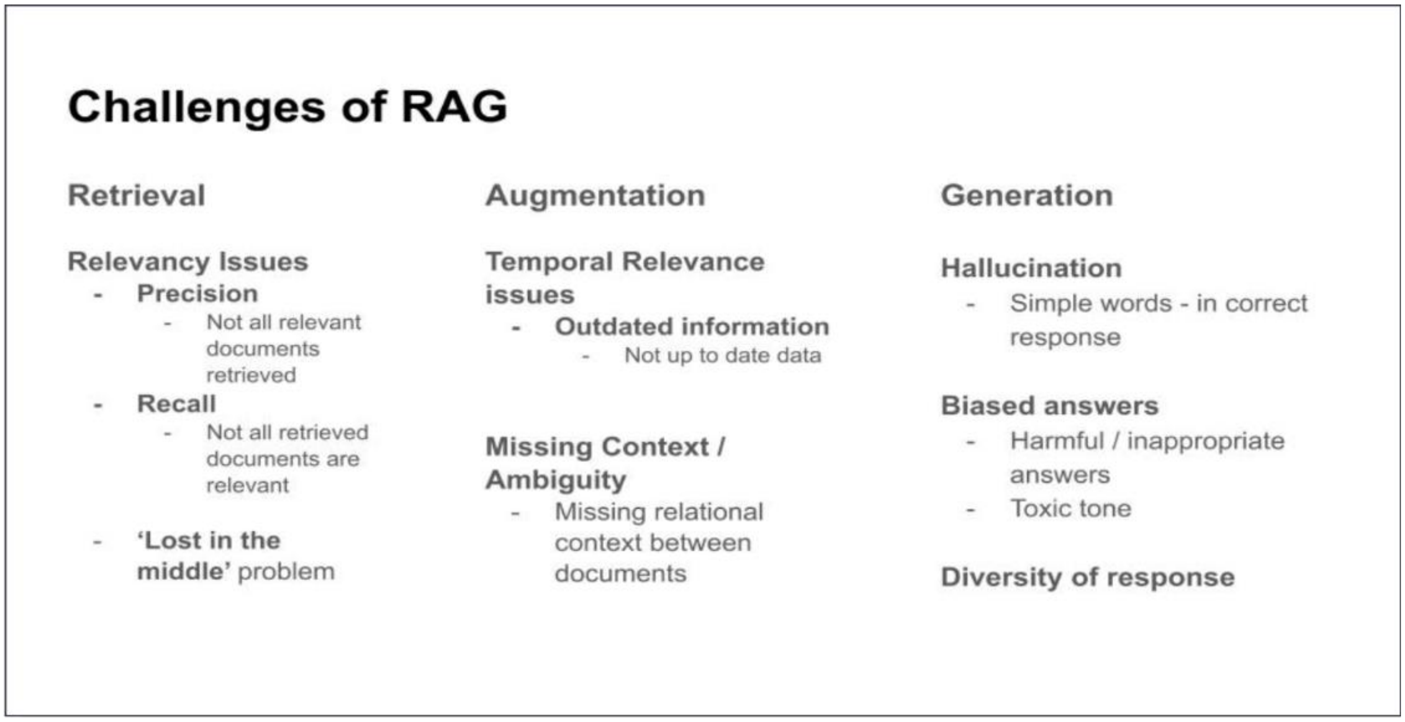
```
If you can't quantify it, you can't improve it
```

It is not only important to have good metrics, but that the ability to measure things separately.

```
If you can't retrieve it, you can't generate it.
```

To see where things are going well, can be improved, and where errors may originate, it's important to evaluate each component in isolation. The following figure classifies the RAG's components – along with what needs evaluation in each:

Figure 24. Challenges of RAG



Challenges of RAG

Retrieval

Relevancy Issues
- **Precision**
  - Not all relevant documents retrieved
- **Recall**
  - Not all retrieved documents are relevant
- **'Lost in the middle'** problem

Augmentation

Temporal Relevance issues
- **Outdated information**
  - Not up to date data

Missing Context / Ambiguity
- Missing relational context between documents

Generation

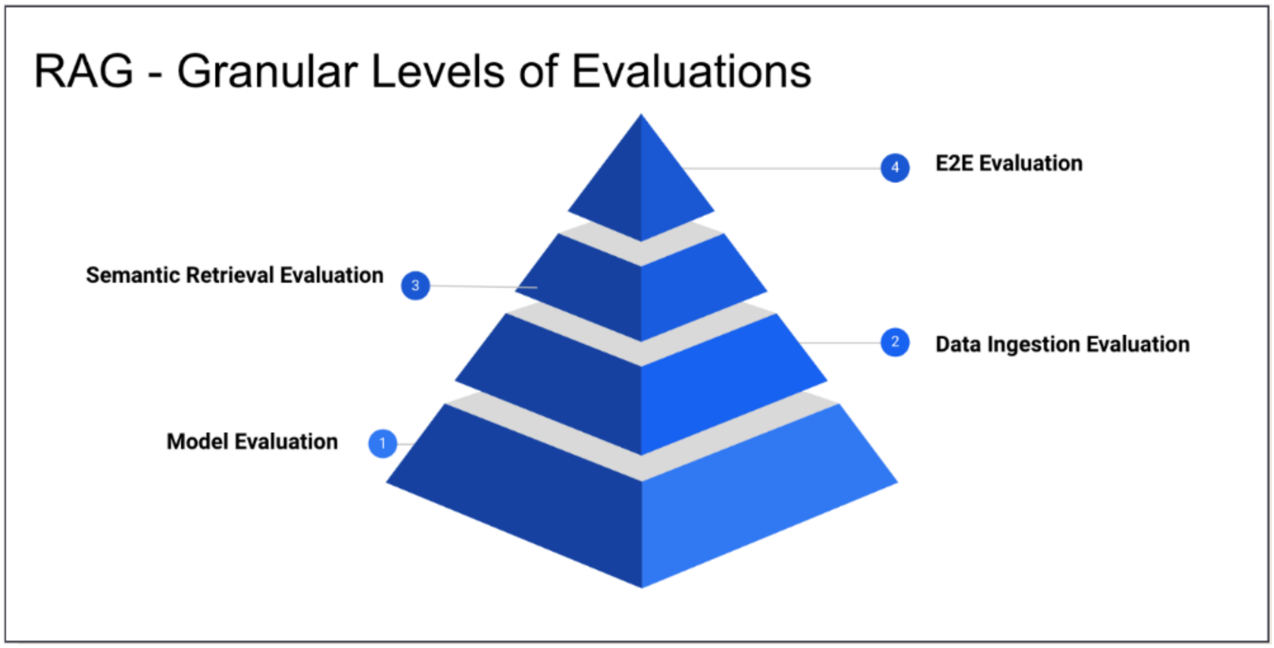
Hallucination
- Simple words - in correct response

Biased answers
- Harmful / inappropriate answers
- Toxic tone

Diversity of response

The Evaluation Framework is meant to ensure granular and thorough measurement, addressing the challenges faced in all three components.

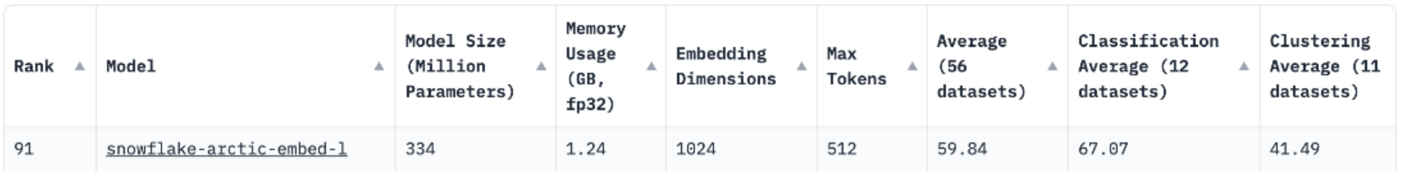
To meet the evaluation challenges systematically, it's a best practice to breakdown evaluation into different levels.


RAG - Granular Levels of Evaluations
4 E2E Evaluation
3 Semantic Retrieval Evaluation
2 Data Ingestion Evaluation
1 Model Evaluation

Embedding Model Evaluation

The Massive Text Embedding Benchmark (MTEB) leverages different public/private datasets to evaluate and report on the different capabilities of individual models. We can use the MTEB to evaluate any model in its list

The Model used in this deployment type is "*snowflake-arctic-embed-l*" and here below is the model's performance listed on leaderboard.



Rank	Model	Model Size (Million Parameters)	Memory Usage (GB, fp32)	Embedding Dimensions	Max Tokens	Average (56 datasets)	Classification Average (12 datasets)	Clustering Average (11 datasets)
91	snowflake-arctic-embed-l	334	1.24	1024	512	59.84	67.07	41.49

Data Ingestion Evaluation

After evaluating model's performance using benchmarks, and (optionally) fine-tune it, then configure data ingestion into semantic retrieval store (vector store).

To evaluate data ingestion, observe and measure how changes in related variables affect ingestion outcomes:

- Chunk Size: The size of each data segment, which depends on the token limit of the embedding model. Chunk size substantially determines data granularity and the contextual understanding of the data, which impacts the precision, recall, and relevancy of results.
- Chunk Overlap: The extent to which data points of events are shared by adjacent data chunks. Overlap helps with retention of context information contained in chunks but should be paired with strategies like deduplication and content normalization to eradicate adverse effects (redundancy or inconsistency).
- Chunking/ Text Splitting Strategy: The process of data splitting and further treatment, based on both data type (for example, html, markdown, code, or pdf, and so on) and nuances of your use-case.

Semantic Retrieval Evaluation

In this deployment, the Milvus Vector Database has been deployed, and milvus offers similarity metrics.

Similarity metrics are used to measure similarities among vectors. Choosing a good distance metric helps improve the classification and clustering performance significantly.

Similarity Metrics	Index Type
• Euclidean distance (L2) • Inner Product (IP)	• FLAT • IVF_FLAT • IVF_SQ8 • IVF_PQ • HNSW • IVF_HNSW • RHNSW_FLAT

Similarity Metrics	Index Type
	• RHNSW_SQ • RHNSW_PQ • ANNOY

Euclidean distance (L2)—essentially, Euclidean distance measures the length of a segment that connects 2 points.

The formula for Euclidean distance is as follows:

$$d(\mathbf{a},\mathbf{b}) = d(\mathbf{b},\mathbf{a}) = \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$$

where a = (a1, a2,..., an) and b = (b1, b2,..., bn) are two points in n-dimensional Euclidean space

It’s the most used distance metric and is very useful when the data are continuous.

Inner Product (IP)—The IP distance between two embeddings is defined as follows:

$$p(A,B) = A \cdot B = \sum_{i=1}^{n} a_i \times b_i$$

ip

Where A and B are embeddings, ||A|| and ||B|| are the norms of A and B.

IP is more useful if you are more interested in measuring the orientation but not the magnitude of the vectors.

If you use IP to calculate embeddings similarities, you must normalize your embeddings. After normalization, the inner product equals cosine similarity.

Suppose X’ is normalized from embedding X:

$$X' = (x'_1, x'_2, \ldots, x'_n), X' \in {}^{n}$$

The correlation between the two embeddings is as follows:

$$x'_i = \frac{x_i}{\| X \|} = \frac{x_i}{\sqrt{\sum_{i=1}^{n} (x_i)^2}}$$

End-to-End Evaluation

An end-to-end evaluation of a RAG application assesses the final outputs generated by LLMs in response to given inputs. It requires addressing issues discussed above related to data heterogeneity, domain specificity, and user query and preference diversity. It's impossible to devise a fixed metric or methodology that fits all domains and use-cases.

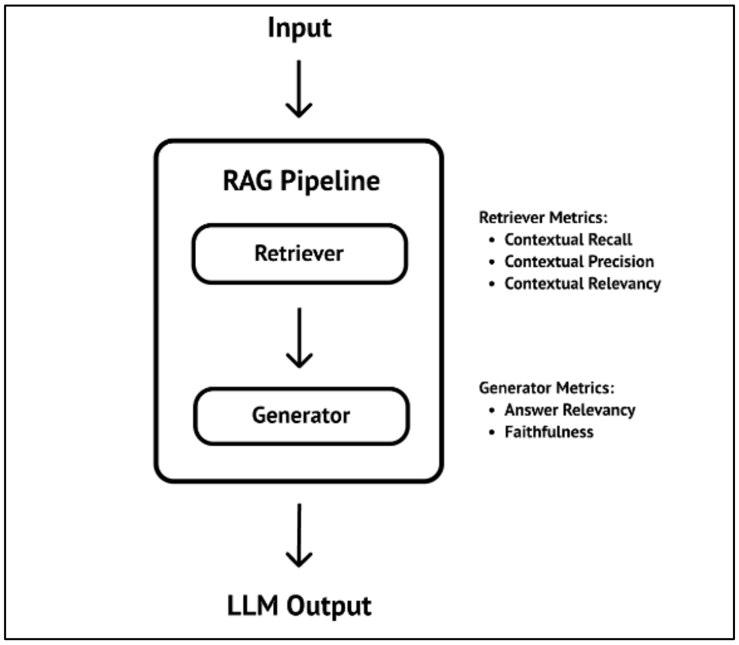
E2E evaluation frameworks range from proprietary solutions to open-source tools. Selecting the right solution requires balancing considerations around ease of maintenance and operational burden, plus how well the metrics observed by the tool map to your Retrieval Augmented Generation pipeline's use case.

Table 16. Recommended RAG Frameworks based on Use Case

Use Case	Framework	Metrics Used	Reasoning
Initial RAG Evaluations	RAGAS	Average Precision (AP), Faithfulness	RAGAS is ideal for initial evaluations, especially in environments where reference data is scarce. It focuses on precision and how faithfully the response matches the provided context.
Dynamic, Continuous RAG Deployments	ARES	MRR, NDCG	ARES uses synthetic data and LLM judges, which are suitable for environments needing continuous updates and training and focusing on response ranking and relevance.
Full System Traces, Including LLM's and Vector Storage	TraceLoop	Informational Gain, Factual Consistency, Citation Accuracy	TraceLoop is best suited for applications where tracing the flow and provenance of information used in the generated output is critical, such as academic research or journalism.
Realtime RAG Monitoring	Arize	Precision, Recall, F1	Arize excels in real-time performance monitoring, making it perfect for deployments where immediate feedback on RAG

Use Case	Framework	Metrics Used	Reasoning
			performance is essential
Enterprise Level RAG Applications	Galileo	Custom Metrics, Context Adherence	Galileo provides advanced insights and metrics integration for complex applications, ensuring RAG's adherence to context.
Optimizing RAG for Specific Domains	TruLens	Domain-Specific Accuracy, Precision	TruLens is designed to optimize RAG systems within specific domains, by enhancing the accuracy and precision of domain-relevant responses

Since a satisfactory LLM output depends entirely on the quality of the retriever and generator, RAG evaluation focuses on evaluating the retriever and generator in RAG pipeline separately. This allows for easier debugging and to pinpoint issues on a component level.



This document provides the steps necessary to run RAGAS Evaluation Framework. RAGAS is a framework that helps evaluate Retrieval Augmented Generation (RAG) Initial pipelines.

Generate Synthetic Dataset

To set up RAGAS evaluation Framework, apply ground truth concept. Some ground truth data - a golden set - provides a meaningful context for the metrics to evaluate our RAG pipeline's generated responses.

The first step in setting up RAGAS is creating an evaluation dataset, complete with questions, answers, and ground-truth data, which takes account of relevant context.

Procedure 1. Create an evaluation dataset

Step 1. Create LLM Prompt template:

```
LLM_PROMPT_TEMPLATE = (
    "<s>[INST] <<SYS>>"
    "{system_prompt}"
    "<</SYS>>"
    " "
    "[The Start of the Reference Context]"
    "{ctx_ref}"
    "[The End of Reference Context][/INST]"
)
```

Step 2. Create the System template:

```
SYS_PROMPT = """
    Given the context paragraph, create two very good question answer pairs.
    Your output should be strictly in a json format of individual question answer pairs with keys from
["question","answer"].
    Restrict the question to the context information provided.
    """
```

Step 3. Generate Synthetic Data:

```
def generate_synthetic_data(
    dataset_folder_path,
    qa_generation_file_path,
    text_splitter_params={"chunk_size": 3000, "chunk_overlap": 100},
):
    files=[dataset_folder_path]
    llm = ChatNVIDIA(base_url="http://10.102.2.216:31641/v1",
model="meta/llama3-8b-instruct",nvidia_api_key="not-used", max_tokens=1024)
    json_data = []
    i = 0
    for pdf_file in files:
        i += 1
        try:
            logger.info(f"{i}/{len(files)}")
            # pdf_file = dataset_folder_path +'/'+ pdf_file
            loader = PyPDFLoader(pdf_file)
            data = loader.load()
            text_splitter = RecursiveCharacterTextSplitter(**text_splitter_params)
            all_splits = text_splitter.split_documents(data)
            for split in all_splits:
                context = LLM_PROMPT_TEMPLATE.format(
                        system_prompt=SYS_PROMPT,
                        ctx_ref=split.page_content)
                try:
                    answer = llm.invoke(context).content
                    question_pattern = r'"question":\s*"([^"]*)"'
                    answer_pattern = r'"answer":\s*"([^"]*)"'
                    question_match = re.findall(question_pattern, answer)
                    answer_match = re.findall(answer_pattern, answer)
                    if(len(question_match)==len(answer_match)):
                        for j, _ in enumerate(question_match):
                            my_data = {
                                'question': question_match[j],
                                'ground_truth_answer': answer_match[j],
                                'ground_truth_context': split.page_content,
                                'document': pdf_file,
                            }
                            json_data.append(my_data)

                except Exception as e:
                    logger.info(
                        f"\n PDF: {pdf_file} \n \t Context: {context} \n Exception Occured: {e}"
                    )

        except Exception as e:
            logger.info(f"\n PDF: {pdf_file} \n Exception Occured: {e}")
    with open(qa_generation_file_path, "w", encoding="utf-8") as f:
        json.dump(json_data, f)
```

Step 4. Call this function with path to the document and file name for q&a.

```
generate_synthetic_data('/home/admin/RAG/flexpod_ai_generative_ocp_m7.pdf','/home/admin/RAG/flexpod_qa_genera
tion.json')
```

This will create a JSON file called “flexpod_qa_generation.json”

Procedure 2. Generate RAG Outputs for the Questions from Synthetic Dataset

Step 1. Load the JSON file and iterate through each question, retrieve context from vector store and ask LLM to generate the response:

```
url_upload = f"http://10.102.2.216:31935/documents"
url_generate = f"http://10.102.2.216:31935/generate"
url_doc_search = f"http://10.102.2.216:31935/search"
f = open('/home/admin/RAG/flexpod_qa_generation.json')
data = json.load(f)

generate_api_params={"use_knowledge_base": True, "temperature":0.2,"top_p":0.7,"max_tokens": 256}

document_search_api_params={"num_docs": 1}

new_data=[]

for entry in data:
    entry_generate = {
        "messages":[
            {
                "role":"user",
                "content":entry["question"]
            }
        ],
        "use_knowledge_base": generate_api_params["use_knowledge_base"],
        "temperature": generate_api_params["temperature"],
        "top_p": generate_api_params["top_p"],
        "max_tokens": generate_api_params["max_tokens"],
        "stop":[
                "string"
        ]
    }
    entry["answer"] = ""
    try:
        with requests.post(url_generate, stream=True, json=entry_generate) as r:
            for chunk in r.iter_lines():
                raw_resp = chunk.decode("UTF-8")
                if not raw_resp:
                    continue
                resp_dict = None
                try:
                    print(raw_resp)
                    resp_dict = json.loads(raw_resp[6:])
                    resp_choices = resp_dict.get("choices", [])
                    if len(resp_choices):
                        resp_str = resp_choices[0].get("message", {}).get("content", "")
                        entry["answer"] += resp_str
                except Exception as e:
                    print(f"Exception Occured: {e}")
    except Exception as e:
        print(f"Exception Occured: {e}")
        entry["answer"] = "Answer couldn't be generated."
    print(entry["answer"])
    entry_doc_search = {
            "query": entry["question"],
            "top_k": document_search_api_params["num_docs"]
        }
    response = requests.post(url_doc_search, json=entry_doc_search).json()
    context_list =typing.cast(typing.List[typing.Dict[str, typing.Union[str, float]]], response)
    contexts = [context.get("content") for context in context_list['chunks']]
    try:
        entry["contexts"] = [contexts[0]]
    except Exception as e:
        print(f"Exception Occured: {e}")
        entry["contexts"] = ""
    new_data.append(entry)
```

Step 2. Once the new list of objects is created, store them on the file system as shown below:

```
with open('/home/admin/RAG/flexpod_eval.json', 'w') as f:
    json.dump(data, f)
```

This creates a JSON file called “flexpod_eval.json”

Procedure 3. Evaluate using the Generated Dataset

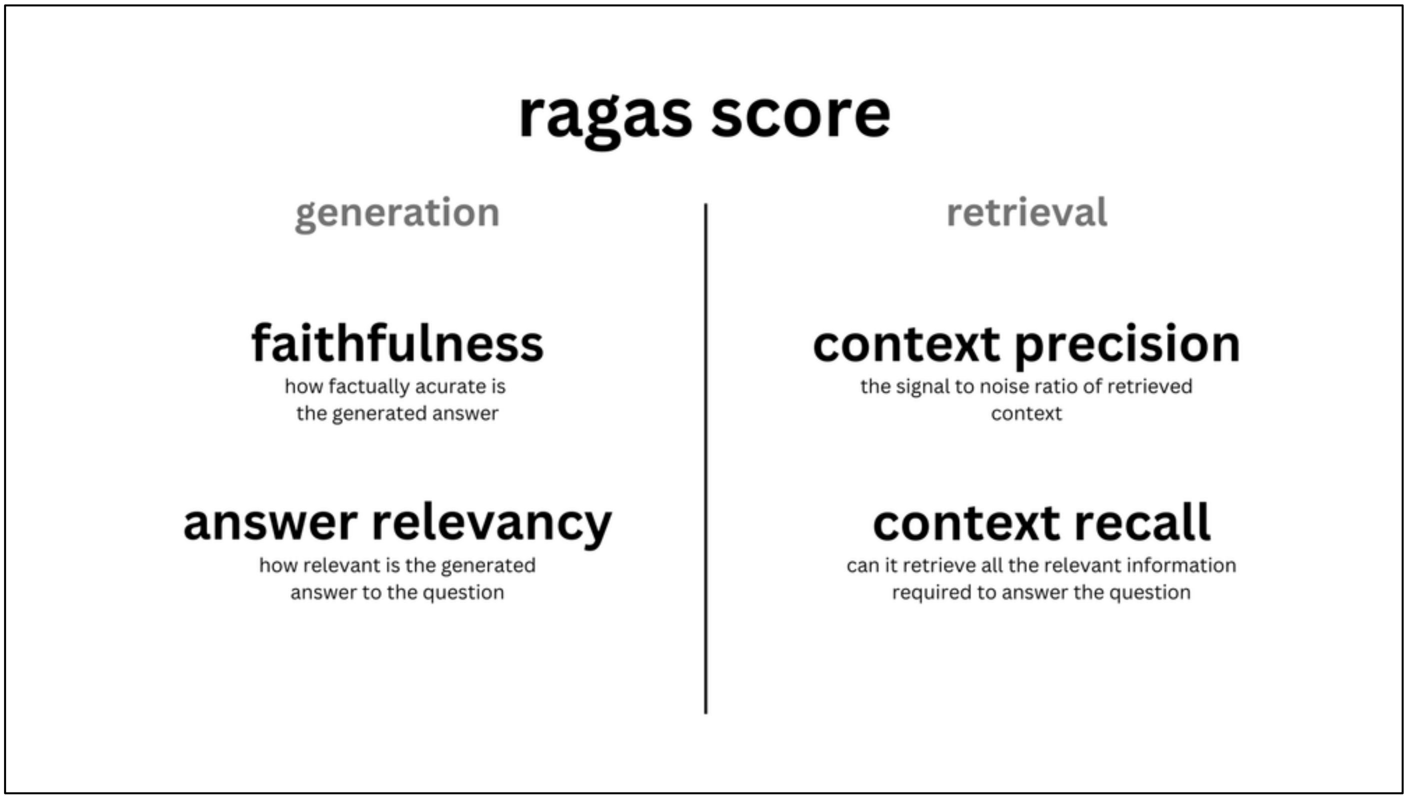
Step 1. Create the Prompt Template:

```
LLAMA_PROMPT_TEMPLATE = (
    "<s>[INST] <<SYS>>"
    "{system_prompt}"
    "<</SYS>>"
    ""
    "Example 1:"
    "[Question]"
    "When did Queen Elizabeth II die?"
    "[The Start of the Reference Context]"
    """On 8 September 2022, Buckingham Palace released a statement which read: "Following further evaluation this
morning, the Queen's doctors are concerned for Her Majesty's health and have recommended she remain under medical
supervision. The Queen remains comfortable and at Balmoral."[257][258] Her immediate family rushed to Balmoral
to be by her side.[259][260] She died peacefully at 15:10 BST at the age of 96, with two of her children, Charles
and Anne, by her side;[261][262] Charles immediately succeeded as monarch. Her death was announced to the public
at 18:30,[263][264] setting in motion Operation London Bridge and, because she died in Scotland, Operation
Unicorn.[265][266] Elizabeth was the first monarch to die in Scotland since James V in 1542.[267] Her death
certificate recorded her cause of death as old age"""
    "[The End of Reference Context]"
    "[The Start of the Reference Answer]"
    "Queen Elizabeth II died on September 8, 2022."
    "[The End of Reference Answer]"
    "[The Start of the Assistant's Answer]"
    "She died on September 8, 2022"
    "[The End of Assistant's Answer]"
    '"Rating": 5, "Explanation": "The answer is helpful, relevant, accurate, and concise. It matches the information
provided in the reference context and answer."'
    ""
    "Example 2:"
    "[Question]"
    "When did Queen Elizabeth II die?"
    "[The Start of the Reference Context]"
    """On 8 September 2022, Buckingham Palace released a statement which read: "Following further evaluation this
morning, the Queen's doctors are concerned for Her Majesty's health and have recommended she remain under medical
supervision. The Queen remains comfortable and at Balmoral."[257][258] Her immediate family rushed to Balmoral
to be by her side.[259][260] She died peacefully at 15:10 BST at the age of 96, with two of her children, Charles
and Anne, by her side;[261][262] Charles immediately succeeded as monarch. Her death was announced to the public
at 18:30,[263][264] setting in motion Operation London Bridge and, because she died in Scotland, Operation
Unicorn.[265][266] Elizabeth was the first monarch to die in Scotland since James V in 1542.[267] Her death
certificate recorded her cause of death as old age"""
    "[The End of Reference Context]"
    "[The Start of the Reference Answer]"
    "Queen Elizabeth II died on September 8, 2022."
    "[The End of Reference Answer]"
    "[The Start of the Assistant's Answer]"
    "Queen Elizabeth II was the longest reigning monarch of the United Kingdom and the Commonwealth."
    "[The End of Assistant's Answer]"
    '"Rating": 1, "Explanation": "The answer is not helpful or relevant. It does not answer the question and instead
goes off topic."'
    ""
    "Follow the exact same format as above. Put Rating first and Explanation second. Rating must be between 1 and
5. What is the rating and explanation for the following assistant's answer"
    "Rating and Explanation should be in JSON format"
    "[Question]"
    "{question}"
    "[The Start of the Reference Context]"
    "{ctx_ref}"
    "[The End of Reference Context]"
    "[The Start of the Reference Answer]"
    "{answer_ref}"
    "[The End of Reference Answer]"
    "[The Start of the Assistant's Answer]"
    "{answer}"
    "[The End of Assistant's Answer][/INST]"
)
```

Step 2. Create System Prompt:

```
SYS_PROMPT = """
    You are an impartial judge that evaluates the quality of an assistant's answer to the question provided.
    You evaluation takes into account helpfullness, relevancy, accuracy, and level of detail of the answer.
    You must use both the reference context and reference answer to guide your evaluation.
    """
```

The performance of individual components within the LLM and RAG pipeline has a significant impact on the overall experience. Ragas offers metrics tailored for evaluating each component of RAG pipeline in isolation.


ragas score
generation
retrieval
faithfulness
how factually acurate is the generated answer
answer relevancy
how relevant is the generated answer to the question
context precision
the signal to noise ratio of retrieved context
context recall
can it retrieve all the relevant information required to answer the question

Step 3. Create a Function to choose the metric types:

```
def calculate_ragas_score(row):
    values = row[['faithfulness', 'context_relevancy', 'answer_relevancy','context_recall']].values
    return statistics.harmonic_mean(values)
```

Step 4. Create a Function to run RAGAS Evaluation from the evaluation json dataset:

```
def eval_ragas(ev_file_path, ev_result_path,llm_model='ai-mixtral-8x7b-instruct'):
    """
    This function evaluates a language model's performance using a dataset and metrics.
    It sets the NVAPI_KEY, initializes a ChatNVIDIA model and LangchainLLM object, loads the
    evaluation dataset, prepares data samples, creates a Dataset object, sets the language model
    for each metric, and evaluates the model with the specified metrics, printing the results.
    """
    llm = ChatNVIDIA(base_url="http://10.102.2.216:31641/v1",
model="meta/llama3-8b-instruct",nvidia_api_key="not-used", max_tokens=1024)
    nvpl_llm = LangchainLLMWrapper(langchain_llm=llm)
    model = "NV-Embed-QA"
    url="http://10.102.2.216:32128/v1/embeddings"
    batch_size=32
    embeddings = NeMoEmbeddings(
    batch_size=batch_size, model=model, api_endpoint_url=url
        )
    nvpl_embeddings = LangchainEmbeddingsWrapper(embeddings)
    try:
        with open(ev_file_path, "r", encoding="utf-8") as file:
            json_data = json.load(file)
    except Exception as e:
        logger.info(f"Error Occured while loading file : {e}")
    eval_questions = []
    eval_answers = []
    ground_truth = []
    vdb_contexts = []
    for entry in json_data:
        eval_questions.append(entry["question"])
        eval_answers.append(entry["answer"])
        vdb_contexts.append(entry['contexts'])
        ground_truth.append(entry["ground_truth_answer"])

    data_samples = {
            'question': eval_questions,
            'answer': eval_answers,
            'contexts': vdb_contexts,
            'ground_truth': ground_truth,
        }
    # print(data_samples)
    dataset = Dataset.from_dict(data_samples)
    print(dataset)
    result = evaluate(
        dataset,
        llm=llm,
        embeddings=nvpl_embeddings,
        metrics=[
            answer_similarity,
            faithfulness,
            context_precision,
            context_relevancy,
            answer_relevancy,
            context_recall
        ],
    )
    df = result.to_pandas()
    df['ragas_score']=df.apply(calculate_ragas_score,axis=1)
    df.to_parquet(ev_result_path+'.parquet')
    result['ragas_score']= statistics.harmonic_mean([result['faithfulness'], result['context_relevancy'],
result['answer_relevancy'], result['context_recall']])
    with open(ev_result_path+'.json', "w", encoding="utf-8") as json_file:
        json.dump(result, json_file, indent=2)
```

The following are the results:

```
eval_ragas('/home/admin/RAG/flexpod_eval.json','/home/admin/RAG/results/flexpod_evaluator')
{
  "answer_similarity": 0.7552602351824748,
  "faithfulness": 0.7331417379645726,
  "context_precision": 0.5098039215176471,
  "context_relevancy": 0.24522045977793314,
  "answer_relevancy": 0.6320471238774731,
  "context_recall": 0.5671783662960134,
  "ragas_score": 0.45520601087765455
}
```

Understanding the Metrics

- Answer Similarity

  The concept of Answer Semantic Similarity pertains to the assessment of the semantic resemblance between the generated answer and the ground truth. This evaluation is based on the ground truth and the answer, with values falling within the range of 0 to 1. A higher score signifies a better alignment between the generated answer and the ground truth.

Measuring the semantic similarity between answers can offer valuable insights into the quality of the generated response. This evaluation utilizes a cross-encoder model to calculate the semantic similarity score.

- Faithfulness

  This measures the factual consistency of the generated answer against the given context. It is calculated from answer and retrieved context. The answer is scaled to (0,1) range. Higher the better.

  The generated answer is regarded as faithful if all the claims made in the answer can be inferred from the given context. To calculate this, a set of claims from the generated answer is first identified. Each of these claims is cross-checked with the given context to determine if it can be inferred from the context. The faithfulness score is determined by:

$$\text{Faithfulness score} = \frac{|\text{Number of claims in the generated answer that can be inferred from given context}|}{|\text{Total number of claims in the generated answer}|}$$

- Context Precision

  Context Precision is a metric that evaluates whether all the ground-truth relevant items present in the contexts are ranked higher or not. Ideally all the relevant chunks must appear at the top ranks. This metric is computed using the question, ground_truth and the contexts, with values ranging between 0 and 1, where higher scores indicate better precision.

$$\text{Context Precision@K} = \frac{\sum_{k=1}^{K} (\text{Precision@k} \times v_k)}{\text{Total number of relevant items in the top } K \text{ results}}$$

$$\text{Precision@k} = \frac{\text{true positives@k}}{(\text{true positives@k} + \text{false positives@k})}$$

Where K is the total number of chunks in contexts and vk∈ {0,1} is the relevance indicator at rank k.

- Context Relevancy

  This metric gauge the relevancy of the retrieved context, calculated based on both the question and contexts. The values fall within the range of (0, 1), with higher values indicating better relevancy.

  Ideally, the retrieved context should exclusively contain essential information to address the provided query. To compute this, we initially estimate the value of |S| by identifying sentences within the retrieved context that are relevant for answering the given question. The final score is determined by the following formula:

$$\text{context relevancy} = \frac{|S|}{|\text{Total number of sentences in retrived context}|}$$

- Answer Relevancy

  The evaluation metric, Answer Relevancy, focuses on assessing how pertinent the generated answer is to the given prompt. A lower score is assigned to answers that are incomplete or contain redundant information and higher scores indicate better relevancy. This metric is computed using the question, the context, and the answer.

  The Answer Relevancy is defined as the mean cosine similarity of the original question to a number of artificial questions, which were generated (reverse engineered) based on the answer:

$$\text{answer relevancy} = \frac{1}{N}\sum_{i=1}^{N} cos(E_{g_i}, E_o)$$

$$\text{answer relevancy} = \frac{1}{N}\sum_{i=1}^{N} \frac{E_{g_i} \cdot E_o}{\|E_{g_i}\|\|E_o\|}$$

Where:

- Egi is the embedding of the generated question i.
- Eo is the embedding of the original question.
- N is the number of generated questions, which is 3 default.

Note: Even though in practice the score will range between 0 and 1 most of the time, this is not mathematically guaranteed, due to the nature of the cosine similarity ranging from -1 to 1.

- Context Recall

  Context recall measures the extent to which the retrieved context aligns with the annotated answer, treated as the ground truth. It is computed using question, ground truth and the retrieved context, and the values range between 0 and 1, with higher values indicating better performance. To estimate context recall from the ground truth answer, each claim in the ground truth answer is analyzed to determine whether it can be attributed to the retrieved context or not. In an ideal scenario, all claims in the ground truth answer should be attributable to the retrieved context. A reference free version of this is available as context_utilization.

  The formula for calculating context recall is as follows:

$$\text{context recall} = \frac{|\text{GT claims that can be attributed to context}|}{|\text{Number of claims in GT}|}$$

- RAGAS Score

  This metric provides a measure of how well the summary captures the important information from the contexts. The intuition behind this metric is that a good summary shall contain all the important information present in the context (or text).

  First extract a set of important key phrases from the context. These key phrases are then used to generate a set of questions. The answers to these questions are always yes (1) for the context. Ask these questions to the summary and calculate the summarization score as the ratio of correctly answered questions to the total number of questions.

  Compute the question-answer score using the answers, which is a list of 1s and 0s. The question-answer score is then calculated as the ratio of correctly answered questions (answer = 1) to the total number of questions.

$$\text{QA score} = \frac{|\text{correctly answered questions}|}{|\text{total questions}|}$$

Milvus Benchmarking with VectorDBBench

Today, the growth of unstructured data and the rise of AI and LLMs have highlighted vector databases as a crucial component of the infrastructure. As the focus shifts to these tools, enterprises need to assess and select the right one for their business. Vector databases manage unstructured data like images, video, text and so on, using vector embeddings. Vector databases specialize in semantic similarity searches using a machine-learning technique called Approximate Nearest Neighbor (ANN). Vector databases are the vector store for RAG applications.

Milvus is a high-performance, highly scalable vector database that runs efficiently across a wide range of environments. Unstructured data varies in format and carries rich underlying semantics, making it challenging to analyze. To manage this complexity, embeddings are used to convert unstructured data into numerical vectors that capture its essential characteristics. These vectors are then stored in a vector database, enabling fast and scalable searches and analytics. Milvus offers robust data modeling capabilities, enabling you to organize your unstructured or multi-modal data into structured collections. Milvus allocates over 80% of its computing resources to its vector databases and search engine. Given the computational demands of high-performance computing, GPUs emerge as a pivotal element of vector database platform, especially within the vector search domain.

NVIDIA’s latest innovation, the GPU-based graph index CAGRA (CUDA ANNs GRAph-based), represents a significant milestone. With NVIDIA’s assistance, Milvus integrated support for CAGRA in its 2.4 version, marking a significant stride toward overcoming the obstacles of efficient GPU implementation in vector search.

The decision to leverage GPU indexes was primarily motivated by performance considerations. We undertook a comprehensive evaluation of Milvus' performance utilizing the VectorDBBench tool, focusing on CAGRA index and observing key evaluation metrics like QPS (Queries Per Second), latency, and recall.

VectorDBBench is the best performing and cost-effective comparison open-source tool, and it offers more than just benchmark results for popular vector databases and cloud services.

Query Performance Metrics

Assessing the query performance of vector databases typically involves three key metrics: latency, queries per second (QPS), and recall rate.

Latency testing measures the time taken for a single query under serial testing conditions. P99 latency is a commonly used metric representing the duration within which 99% of queries are completed. It offers a more nuanced perspective than average latency and aligns closely with user experience.

Note: While latency testing is straightforward, it's heavily influenced by network conditions, especially for cloud products.

QPS refers to a database's query capability under high concurrency. It is achieved by simultaneously sending multiple requests from the test client to maximize database CPU/ GPU utilization and observe throughput. Unlike latency, QPS is less susceptible to network fluctuations, providing a comprehensive evaluation of a vector database's real-world performance.

Recall is the proportion of the ground truth documents that are represented in the retrieved chunks. This is a measure of the completeness of the results.

$$\text{Recall} = \frac{\text{Number of relevant retrieved items}}{\text{Number of total relevant items}}$$

High recall ensures that the LLM is comprehensive in its responses, capturing all the relevant information necessary for the task.

Procedure 1. Milvus Standalone deployment with NetApp NFS FlexGroup

Step 1. Create a new namespace for milvus:

```
oc create namespace milvus-dbstore
```

Step 2. Add the milvus repository:

```
    helm repo add milvus https://zilliztech.github.io/milvus-helm/
```

Step 3. Update the helm repository:

```
helm repo update
```

Note: Make sure that storageclass created for flexgroup is the default storageclass in the cluster FlexGroup has a requirement of minimum size of the PVC to be 800G. Therefore, update the custom_value.yaml file to update the PVC size.

Step 4. Create a file named custom_value.yaml to ensure PVCs are on flexgroup and use the milvus image for deployment on GPUs:

```
image:
  all:
    repository: milvusdb/milvus
    tag: latest-gpu
    pullPolicy: IfNotPresent

standalone:
  replicas: 1  # Run standalone mode with replication disabled
  resources:
    requests:
      nvidia.com/gpu: "1"
    limits:
      nvidia.com/gpu: "1"

persistentVolumeClaim:
      storageClass:
      accessModes: ReadWriteOnce
      size: 800Gi
```

Step 5. Install the helm chart and point to the above created file using -f argument as shown below:

```
helm install milvus milvus/milvus -f custom_value.yaml -n milvus-dbstore
```

Step 6. Check the status of the pods:

```
oc get pods -n milvus-dbstore
```

All pods should be running and in a ready state within couple of minutes:

```
[admin@ocp-installer ~]$ oc get all -n milvus-dbstore
Warning: apps.openshift.io/v1 DeploymentConfig is deprecated in v4.14+, unavailable in v4.10000+
Warning: kubevirt.io/v1 VirtualMachineInstancePresets is now deprecated and will be removed in v2.
NAME                                         READY   STATUS    RESTARTS   AGE
pod/milvus-db-etcd-0                         1/1     Running   2          10h
pod/milvus-db-minio-74bc764f5c-vtfrb         1/1     Running   2          10h
pod/milvus-db-standalone-5d765d788-bpdcz     1/1     Running   0          8h

NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
 AGE
service/milvus-db                   NodePort    172.30.27.219    <none>        19530:32063/TCP,9091:30467/TCP
 10h
service/milvus-db-etcd              ClusterIP   172.30.121.134   <none>        2379/TCP,2380/TCP
 10h
service/milvus-db-etcd-headless     ClusterIP   None             <none>        2379/TCP,2380/TCP
 10h
service/milvus-db-minio             ClusterIP   172.30.40.200    <none>        9000/TCP
 10h

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/milvus-db-minio        1/1     1            1           10h
deployment.apps/milvus-db-standalone   1/1     1            1           10h

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/milvus-db-minio-74bc764f5c       1         1         1       10h
replicaset.apps/milvus-db-standalone-5d765d788   1         1         1       10h

NAME                              READY   AGE
statefulset.apps/milvus-db-etcd   1/1     10h
```

```
NAME                                        HOST/PORT                                                     PATH   SE
RVICES     PORT     TERMINATION   WILDCARD
route.route.openshift.io/milvus-route   milvus-route-milvus-vectordb.apps.ocp.flexpodb4.cisco.com          mi
lvus-db   milvus                  None
```

The milvus version used in the solution is 2.4.12:

```
[admin@ocp-installer ~]$ helm list -n milvus-dbstore
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART
           APP VERSION
milvus-db       milvus-dbstore 1               2024-10-10 14:23:34.0631291 -0400
EDT    deployed        milvus-4.2.12   2.4.12
```

For detailed information, click the following links:

- https://milvus.io/docs/install_cluster-helm-gpu.md
- https://github.com/milvus-io/milvus

VectorDBBench

VectorDBBench is an open-source benchmarking tool designed for high-performance data storage and retrieval systems. This tool allows users to test and compare different vector database systems' performance to determine their specific use case's most suitable database system. Using VectorDBBench, users can make informed decisions based on the actual vector database performance of the systems they are evaluating.

VectorDBBench is written in Python and licensed under the MIT open-source license, The tool is actively maintained by a community of developers committed to improving its features and performance.

Procedure 1. Install and run the VectorDBBench tool

Step 1. Prerequisite:

```
Python >= 3.11
```

Step 2. Install vectordb-bench:

```
pip install vectordb-bench
```

Step 3. Run vectordb-bench:

```
python -m vectordb_bench
```

OR

```
init_bench
```

After completion, the tool is up and running and can access the streamlit web application via the network URL.

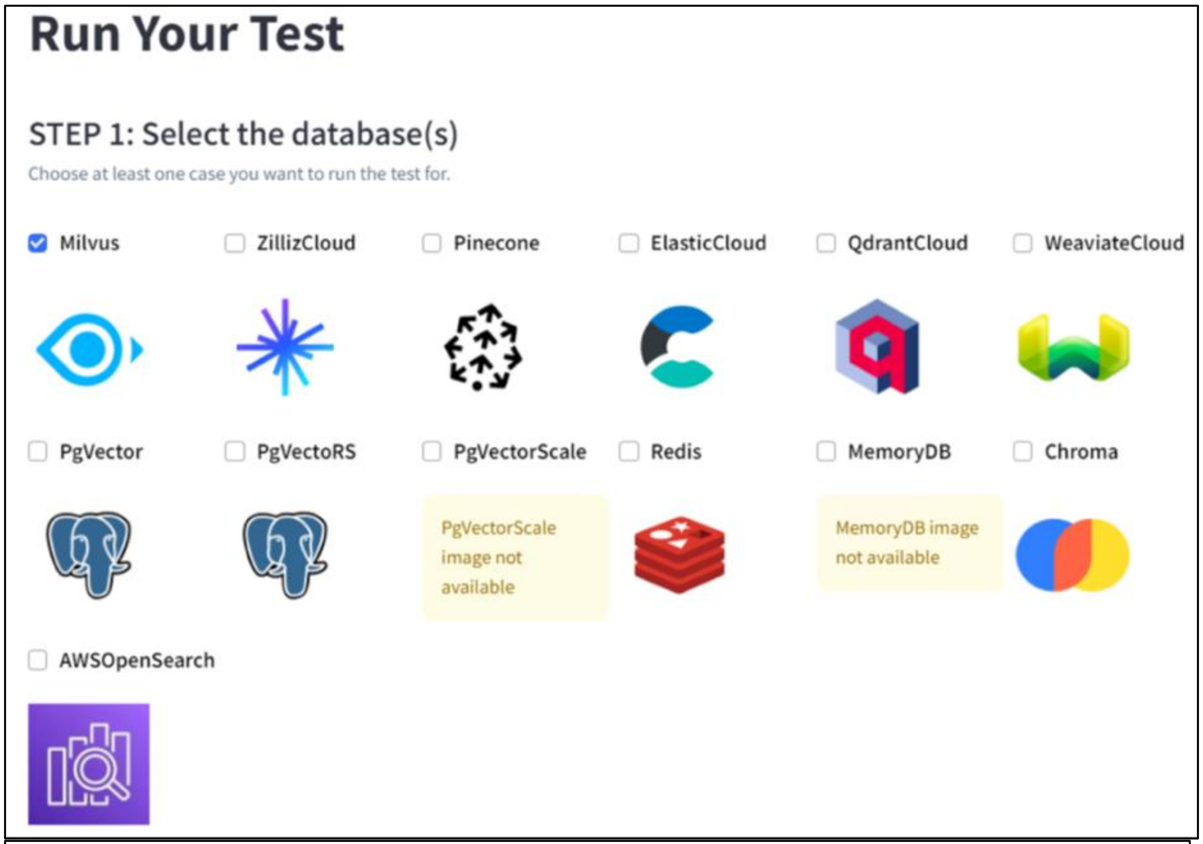
For more information, see: https://github.com/zilliztech/VectorDBBench

Procedure 2. Run the Test

Step 1. Select the database(s). There are various options available for vector databases like Milvus, Pinecone, PgVector, and so on.

Note: In our tests, we used Milvus.

Step 2. Provide the required configuration information for the selected database:



Run Your Test

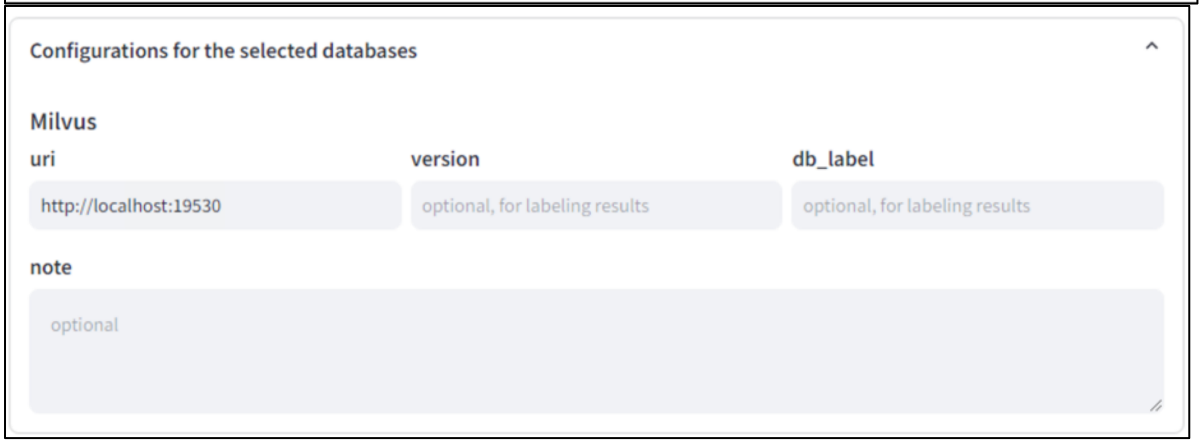
STEP 1: Select the database(s)

Choose at least one case you want to run the test for.

- [✓] Milvus
- [ ] ZillizCloud
- [ ] Pinecone
- [ ] ElasticCloud
- [ ] QdrantCloud
- [ ] WeaviateCloud
- [ ] PgVector
- [ ] PgVectoRS
- [ ] PgVectorScale
- [ ] Redis
- [ ] MemoryDB
- [ ] Chroma
- [ ] AWSOpenSearch

PgVectorScale image not available

MemoryDB image not available



Configurations for the selected databases

Milvus

uri	version	db_label
http://localhost:19530	optional, for labeling results	optional, for labeling results

note

optional

Step 3. Select the test case to run the benchmarking test.

Note: We ran a Search Performance Test case with a medium dataset (Cohere 1M vectors, 768 dimensions)

Step 4. Select the index type. You can select the index type depending upon their deployment characteristics and datasets. Some of the options provided by vectordbbench are DISKANN, HNSW, GPU_IVF_FLAT, GPU_IVF_PQ and so on. There is a flexibility provided to change the test parameters pertaining to a particular index type.

Note: We used GPU_CAGRA index with default parameters for running the tests.

- GPU_CAGRA Index Type

  GPU_CAGRA is a graph-based index optimized for GPUs. Using inference-grade GPUs to run the Milvus GPU version can be more cost-effective compared to using expensive training-grade GPUs.

Note: For more information about index building and search parameters, go to: https://milvus.io/docs/gpu_index.md.

Note: For detailed information about the integration of milvus with CAGRA, go to: https://zilliz.com/blog/Milvus-introduces-GPU-index-CAGRA

- Task Label

  You can provide your custom task label or go by default as provided by the tool.

Benchmarking Test Details

In this solution, we executed Search Performance test of Milvus vectorstore DB with Cohere 1M dataset size and 768 dimensions and varied the batch size for the ML model. To validate the test, we used latencyp99, recall, load-duration, and QPS (Queries Per Second) as the KPIs.

To update the batch size between test iterations, update the NUM_PER_BATCH variable in the __init__.py file which is stored under python3.11/site-packages/vectordb_bench directory.

Note: We used batch sizes of 1, 10, 50, and 100 in the test iterations.

Note: The dataset gets downloaded on the client machine where vectordb_bench is running. Therefore, make sure there is sufficient disk space on the system.

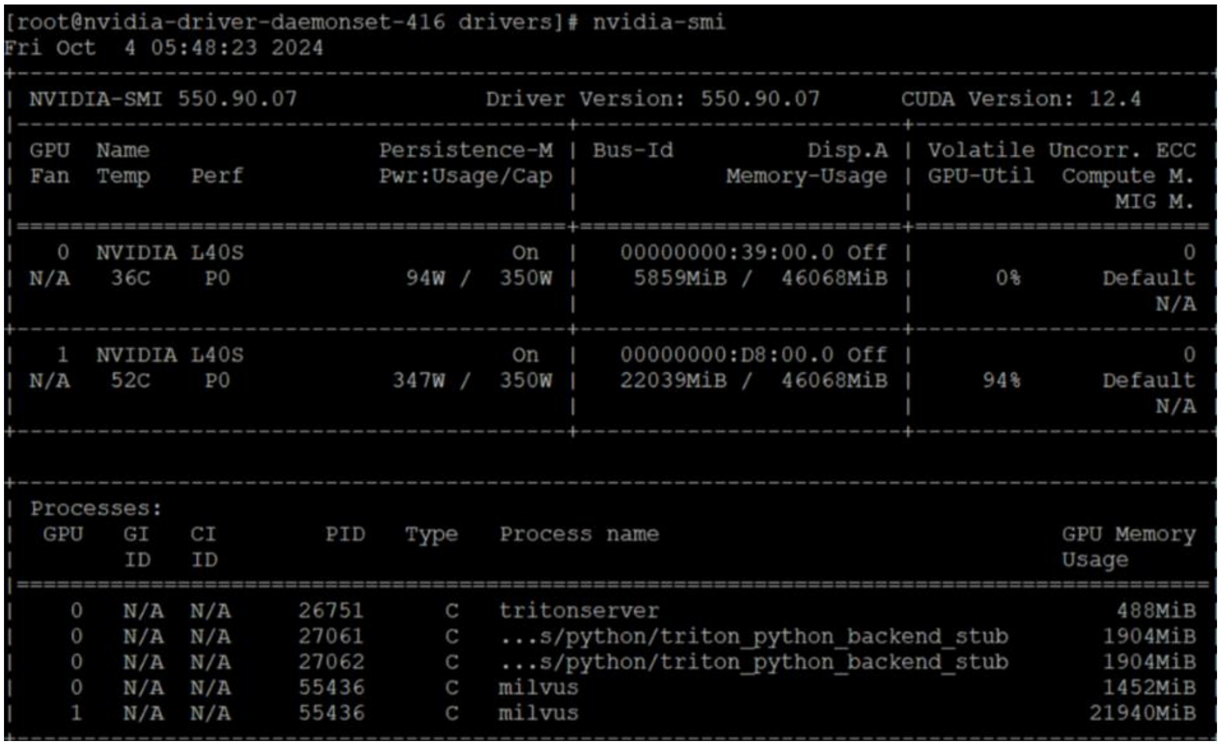
Since these tests are performed with GPU-based index, you will also observe GPU utilization during the test.

Test Results

This chapter contains the following:

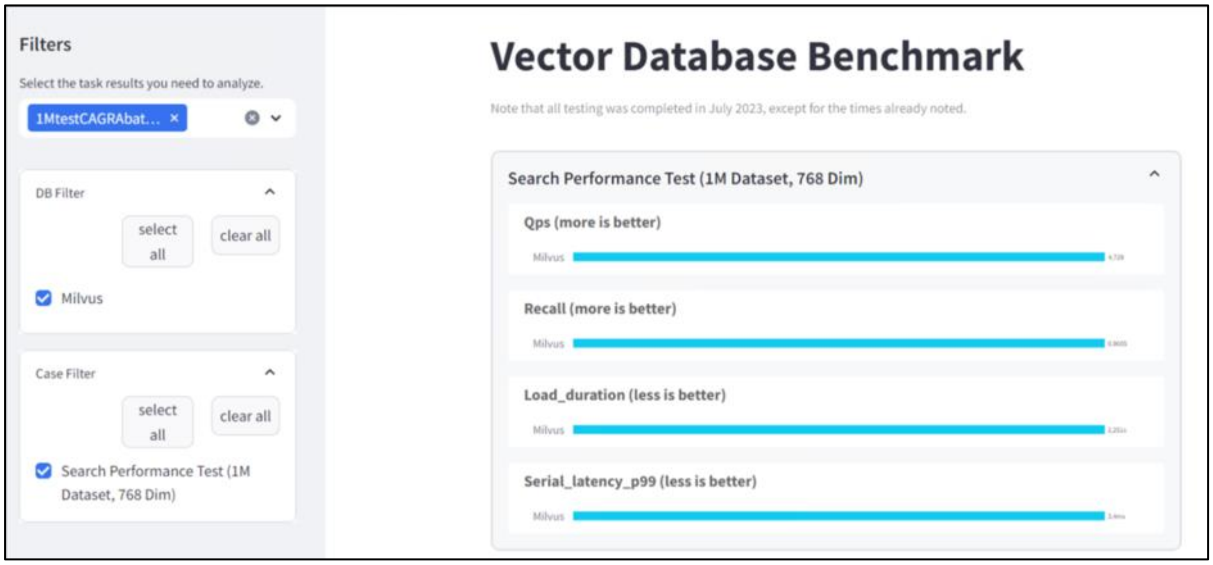
Batch Size 1

While running the test, we noticed that the GPU was being utilized during multiple processes like data ingestion, querying, and so on. One instant is shown below using the **nvidia-smi** tool.



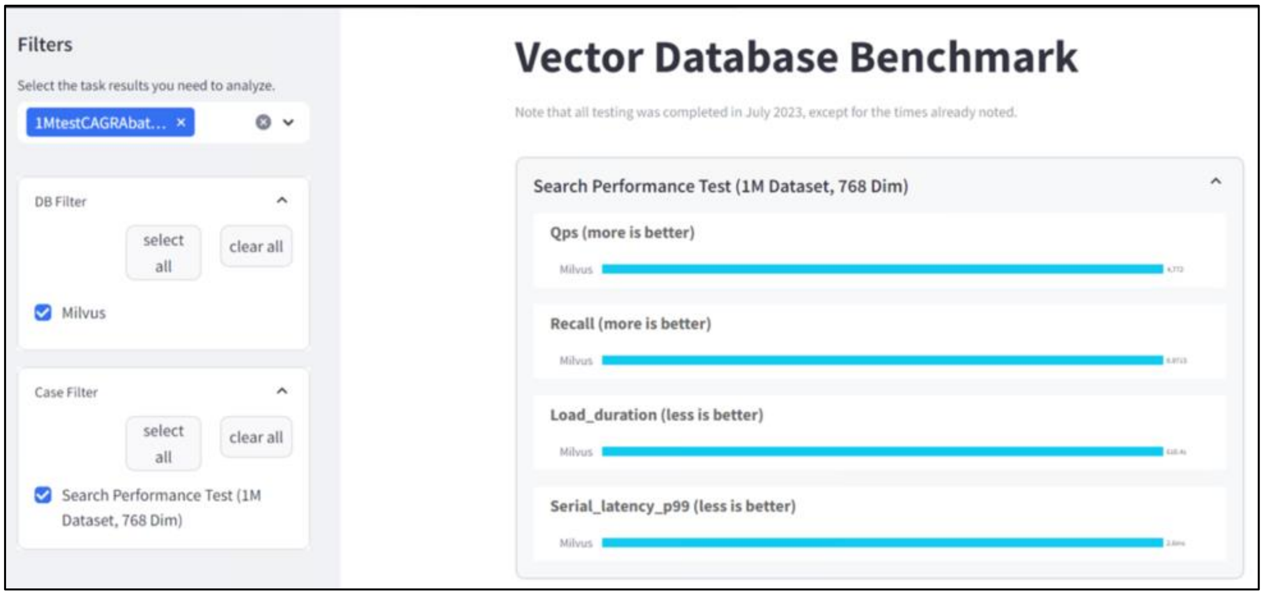
```
[root@nvidia-driver-daemonset-416 drivers]# nvidia-smi
Fri Oct  4 05:48:23 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.90.07              Driver Version: 550.90.07      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA L40S                    On  |   00000000:39:00.0 Off |                    0 |
| N/A   36C    P0             94W /  350W |    5859MiB /  46068MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA L40S                    On  |   00000000:D8:00.0 Off |                    0 |
| N/A   52C    P0            347W /  350W |   22039MiB /  46068MiB |     94%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A     26751      C   tritonserver                                  488MiB |
|    0   N/A  N/A     27061      C   ...s/python/triton_python_backend_stub       1904MiB |
|    0   N/A  N/A     27062      C   ...s/python/triton_python_backend_stub       1904MiB |
|    0   N/A  N/A     55436      C   milvus                                       1452MiB |
|    1   N/A  N/A     55436      C   milvus                                      21940MiB |
+-----------------------------------------------------------------------------------------+
```

With batch size 1, we observed QPS of around 4729, Recall of 0.9605, and latencyp99 of 3.4ms.


Filters
Select the task results you need to analyze.
1MtestCAGRAbat...
DB Filter
select all
clear all
Milvus
Case Filter
select all
clear all
Search Performance Test (1M Dataset, 768 Dim)
Vector Database Benchmark
Note that all testing was completed in July 2023, except for the times already noted.
Search Performance Test (1M Dataset, 768 Dim)
Qps (more is better)
Milvus
4,728
Recall (more is better)
Milvus
0.9605
Load_duration (less is better)
Milvus
1,251s
Serial_latency_p99 (less is better)
Milvus
3.4ms

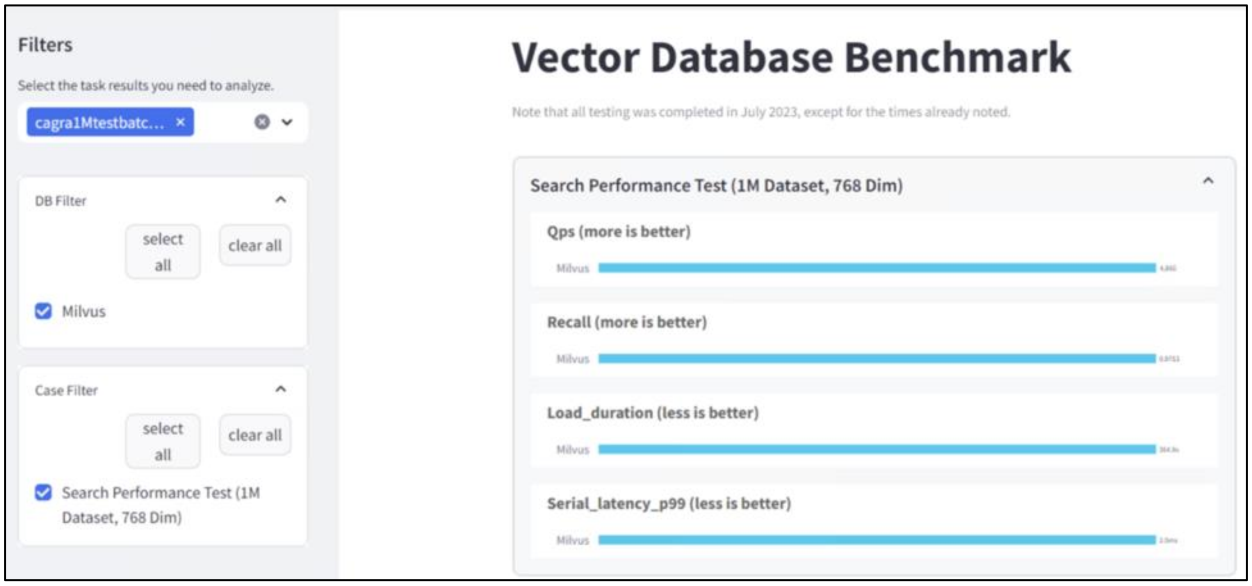
Batch Size 10

With batch size 10, latencyp99 of 2.6ms, recall of 0.9715, and QPS of 4772 was observed.


Filters
Select the task results you need to analyze.
1MtestCAGRAbat...
DB Filter
select all
clear all
Milvus
Case Filter
select all
clear all
Search Performance Test (1M Dataset, 768 Dim)
Vector Database Benchmark
Note that all testing was completed in July 2023, except for the times already noted.
Search Performance Test (1M Dataset, 768 Dim)
Qps (more is better)
Milvus
4,772
Recall (more is better)
Milvus
0.9715
Load_duration (less is better)
Milvus
610.4s
Serial_latency_p99 (less is better)
Milvus
2.6ms

Batch Size 50

With batch size 50, recall of 0.9753, latencyp99 of 2.5ms, and QPS of 4866 was observed.


Filters

Select the task results you need to analyze.

cagra1Mtestbatc... ×

DB Filter

select all | clear all

☑ Milvus

Case Filter

select all | clear all

☑ Search Performance Test (1M Dataset, 768 Dim)

Vector Database Benchmark

Note that all testing was completed in July 2023, except for the times already noted.

Search Performance Test (1M Dataset, 768 Dim)

Qps (more is better)

Milvus 4,846

Recall (more is better)

Milvus 0.9753

Load_duration (less is better)

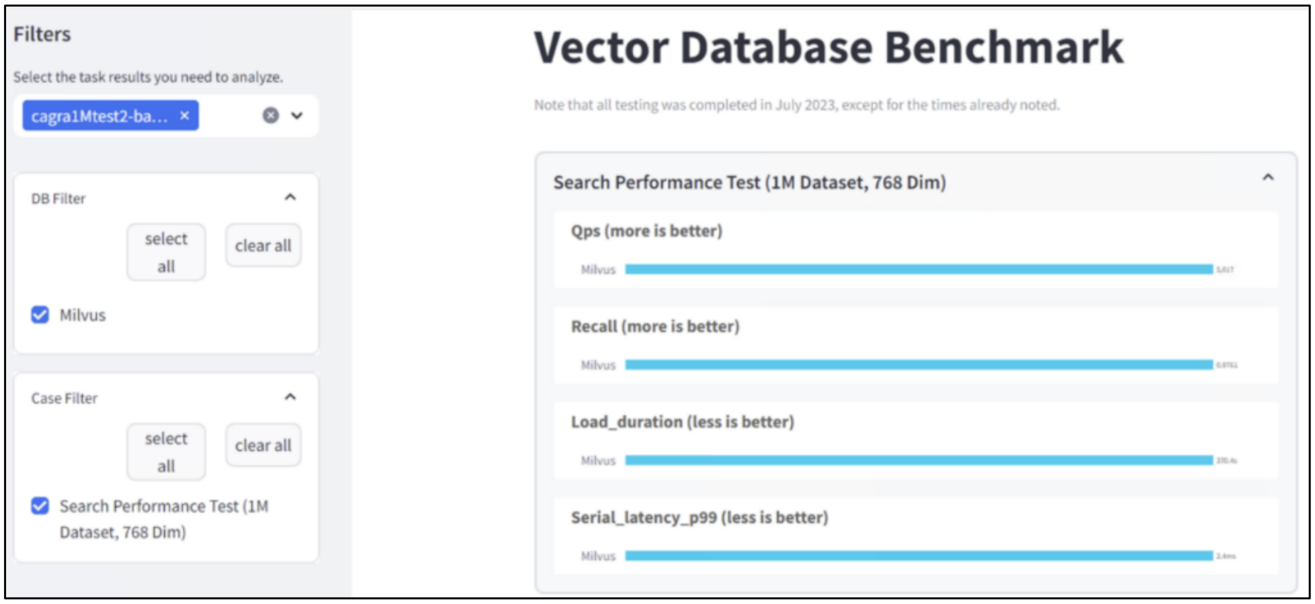
Milvus 364.3s

Serial_latency_p99 (less is better)

Milvus 2.3ms

Batch Size 100

With batch size 100, QPS of 5017, latencyp99 of 2.4ms, and recall of 0.9761 was recorded.


Filters

Select the task results you need to analyze.

cagra1Mtest2-ba... ×

DB Filter

select all | clear all

☑ Milvus

Case Filter

select all | clear all

☑ Search Performance Test (1M Dataset, 768 Dim)

Vector Database Benchmark

Note that all testing was completed in July 2023, except for the times already noted.

Search Performance Test (1M Dataset, 768 Dim)

Qps (more is better)

Milvus 5,017

Recall (more is better)

Milvus 0.9761

Load_duration (less is better)

Milvus 370.4s

Serial_latency_p99 (less is better)

Milvus 2.4ms

Results Inference

Table 17 lists the test results and specifications.

Table 17. Test Results

Test configuration: Vector Database - Milvus standalone Index type - GPU_CAGRA

Storage backend – NFS FlexGroup Cluster details - 2 worker nodes with GPU and 3 master nodes OpenShift version – 4.16

Batch size	Recall	serial_latency_p99 (ms)	QPS (Query per second)
1	0.9605	3.4	4729
10	0.9715	2.6	4772
50	0.9753	2.5	4866
100	0.9761	2.4	5017

From these results, we can infer that as the batch size increases, the Multi Modal RAG performs better, that is validated by the trend in the output KPIs with Milvus vectorstore using VectorDBBench tool.

Note: Our test results are in agreement with the observations made by Zilliz as explained in the following link: https://zilliz.com/blog/Milvus-introduces-GPU-index-CAGRA

The benchmark results underscore the substantial performance benefits of adopting GPU-accelerated indexes like CAGRA in Milvus. Not only does it excel in accelerating search tasks across batch sizes, but it also significantly enhances index construction speed, affirming the value of GPUs in optimizing vector database performance.

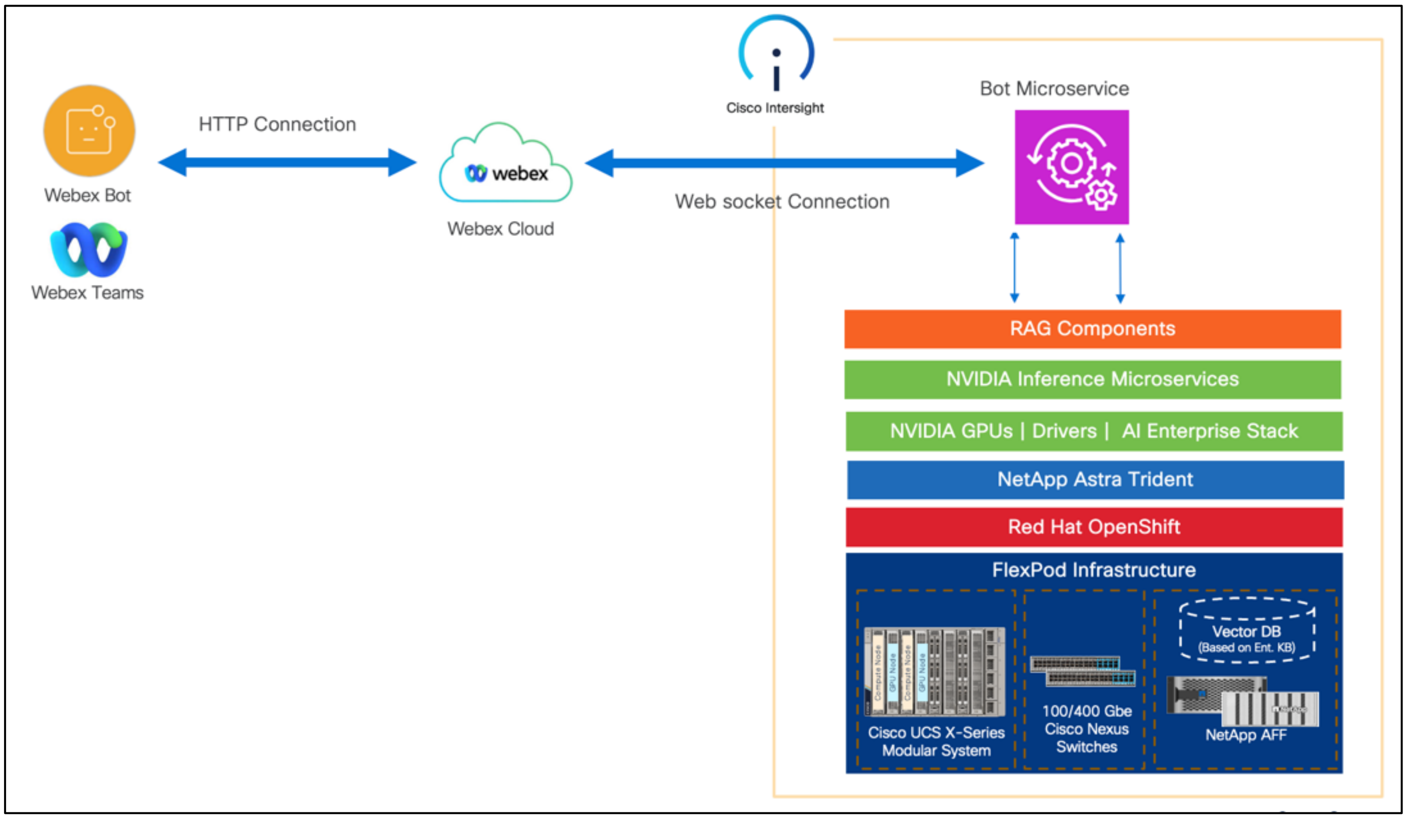
Inferencing with Webex Bot

This chapter contains the following:

About Webex Bot

Webex Bots are like regular Webex users. They can participate in 1-to-1 and group spaces and users can message them directly or add them to a group space. A special badge is added to a bot's avatar in the Webex clients, so users know they're interacting with a bot instead of a human.

In this design, the Webex Bot Microservice is deployed to the stack and that microservice calls the NVIDIA NIM API to perform the LLM Inferencing.



About NVIDIA NIM Microservice

NVIDIA NIM is a set of easy-to-use microservices designed to accelerate the deployment of generative AI models across the cloud, data center, and workstations. NIM packages are categorized by model family and a per model basis. For example, NVIDIA NIM for large language models (LLMs) brings the power of state-of-the-art LLMs to enterprise applications, providing unmatched natural language processing and understanding capabilities.

In this solution, we demonstrated how quickly you can get started with NVIDIA NIM (as a Large Language Model inference framework).

NIM makes it easy for IT and DevOps teams to self-host large language models (LLMs) in their own managed environments while still providing developers with industry standard APIs that allow them to build powerful copilots, chatbots, and AI assistants that can transform their business. Leveraging NVIDIA's cutting-edge GPU acceleration and scalable deployment, NIM offers the fastest path to inference with unparalleled performance.

Deploy BOT Microservice

A prerequisite is to create a Config File JSON and update the bot's name and bot token. To create a new bot, visit developer.Webex.com.

Procedure 1. Deploy BOT microservice

Step 1. Create Bot config.json file:

```
  {
     "bot_name": "examplename…..",
     "Webex_token": "lakjsdflkjalsjfd…………………………",
     "approved_domains": ["cisco.com"],
     "approved_users": [],
     "approved_rooms": [],
     "ai_generate_endpoint": "http://10.102.2.216:32126/api/chain-server/generate",
     "ai_document_upload_url":"http://10.102.2.216:31935/documents",
     "ai_generate":"http://10.102.2.215:31641/v1",
     "ai_model":"meta/llama3-8b-instruct",
     "ai_embed":"http://10.102.2.214:32128/v1",
     "ai_embed_model":"NV-Embed-QA",
     "nvidia_api_key":"not-used",
     "batch_size":32,
     "max_tokens":1024,
     "temperature":0.75,
     "top_p":0.7


}
```

Step 2. Mount this config.json file to bot microservice to start interacting with NVIDIA NIM. Next Section describes more on how to access different services which are being deployed on this stack.

AI Endpoints from NVIDIA NIM

The AI Generate, embedding endpoints are exposed from multiple microservices. Below is the example way to find out the list of endpoints needed for inferencing based on the use case.

Procedure 1. Run Kubernetes or OpenShift commands

Step 1. Fetch the Node Port for Chain-Server, Playground and NIM LLM using the following commands:

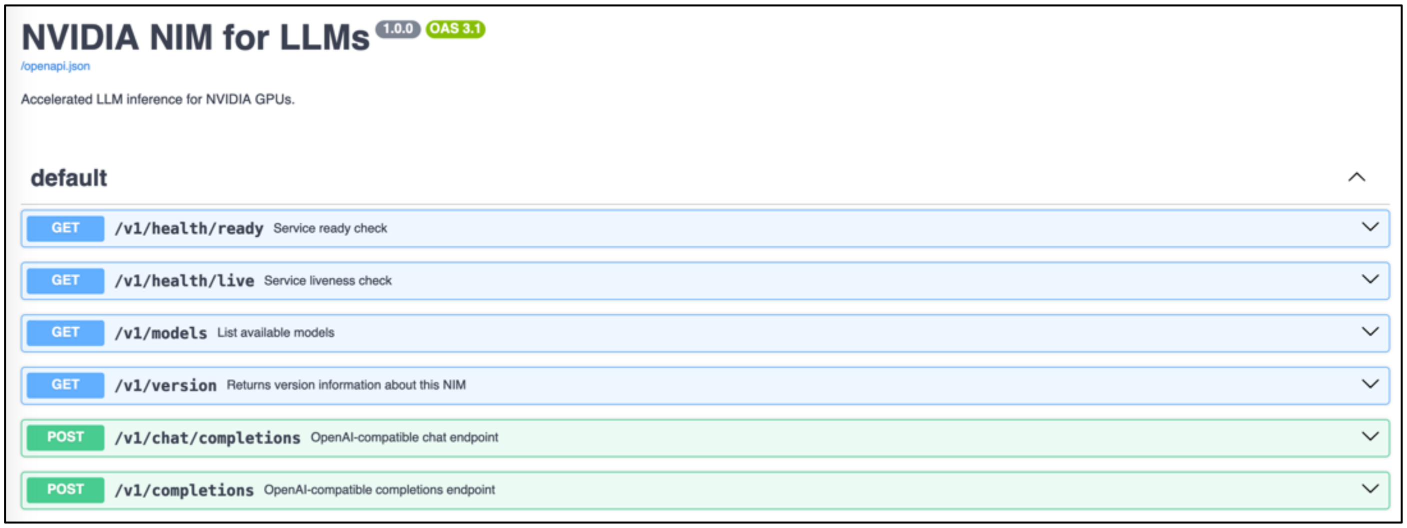
```
[admin@ocp-installer conf]$ kubectl get svc -A | grep 31935
canonical-rag                                       chain-server
NodePort       172.30.72.223    <none>                                    8081:31935/TCP
4d4h
```

```
[admin@ocp-installer conf]$ kubectl get svc -A | grep 32126
canonical-rag                                       rag-playground
NodePort       172.30.214.103   <none>                                    3001:32126/TCP
4d4h
```

```
[admin@ocp-installer conf]$ kubectl get svc -A | grep 31641
nim-llm                                             pkoppa-nim-llm
NodePort       172.30.66.137    <none>                                    8000:31641/TCP
4d8h
```

```
[admin@ocp-installer conf]$ kubectl get svc -A | grep 32128
text-embedding-nim                                  nemo-embedder-nemo-retriever-embedding-ms
NodePort       172.30.14.58     <none>                                    8080:32128/TCP,8001:31785/TCP
4d6h
```

Once the Node Ports are fetched, the API's can be viewed for each microservice. Depending on inferencing requirements, multiple of these API's can be leveraged:


NVIDIA NIM for LLMs 1.0.0 OAS 3.1

/openapi.json

Accelerated LLM inference for NVIDIA GPUs.

default

GET /v1/health/ready Service ready check

GET /v1/health/live Service liveness check

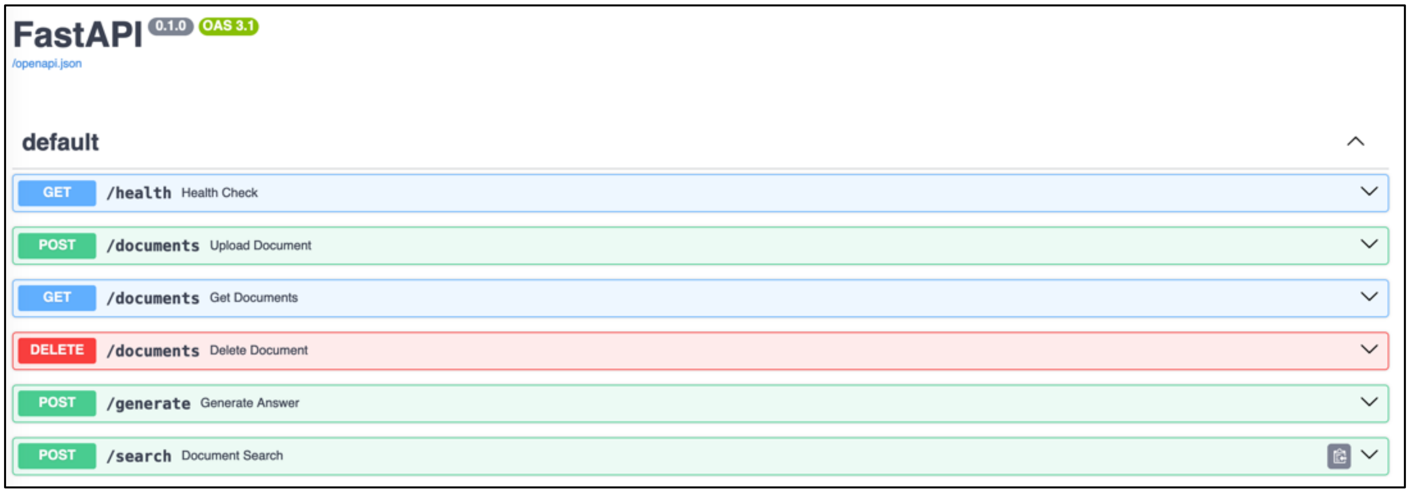
GET /v1/models List available models

GET /v1/version Returns version information about this NIM

POST /v1/chat/completions OpenAI-compatible chat endpoint

POST /v1/completions OpenAI-compatible completions endpoint

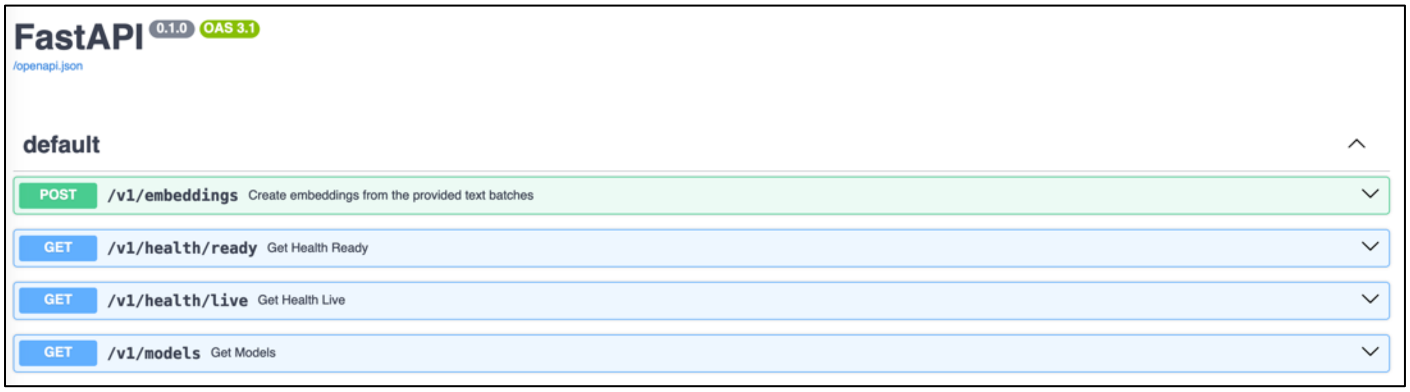
In this solution, the NVIDIA Chain-Server has been deployed to perform RAG Testing. Chain Server provides APIs to upload, delete, Search, and generate answers from Vector Store:


FastAPI 0.1.0 OAS 3.1
/openapi.json

default

GET /health Health Check
POST /documents Upload Document
GET /documents Get Documents
DELETE /documents Delete Document
POST /generate Generate Answer
POST /search Document Search

This solution also contains NVIDIA NeMo Retriever Embedding Microservice and their API’s can be accessed:


FastAPI 0.1.0 OAS 3.1
/openapi.json

default

POST /v1/embeddings Create embeddings from the provided text batches
GET /v1/health/ready Get Health Ready
GET /v1/health/live Get Health Live
GET /v1/models Get Models

Inferencing with NIM

Python based OpenAI libraries or Langchain based ChatNVIDIA Library can be leveraged to generate answers from user’s queries.

Procedure 1. Create LLM Class

Step 1. Call the Generate Endpoint from the Python Class:

```
class LLM:
    def __init__(self):
        self.model = ChatNVIDIA(
            base_url=CONF.ai_generate,
            model=CONF.ai_model,
            nvidia_api_key=CONF.nvidia_api_key,
            max_tokens=CONF.max_tokens
        )

    def __call__(self):
        return self.model


    def _generate(self, query):
        response=self.model.invoke(query)
        return response.content
```

Procedure 2. Classify User Query

Since the users through the Webex bot can ask general text generation queries or RAG based queries, pass the user question through a router, and classify between General or Knowledge.

Step 1. Create the LLM Prompt to classify:

```
init_prompt='''
Given the user question below, classify it as either being about fetching results from the knowledge base or general
enquiry.
example: from the attachment, tell me the number of employees in the company.
classification: knowledge
example: Write a python program to calculate the factorial of a number.
classification: general enquiry
Do not respond with more than one word.

<question>
{question}
</question>

Classification:'''
```

Step 2. Create a general enquiry prompt:

```
general_prompt='''
    Respond to the user question below.
    Question: {question}
    Answer:
'''
```

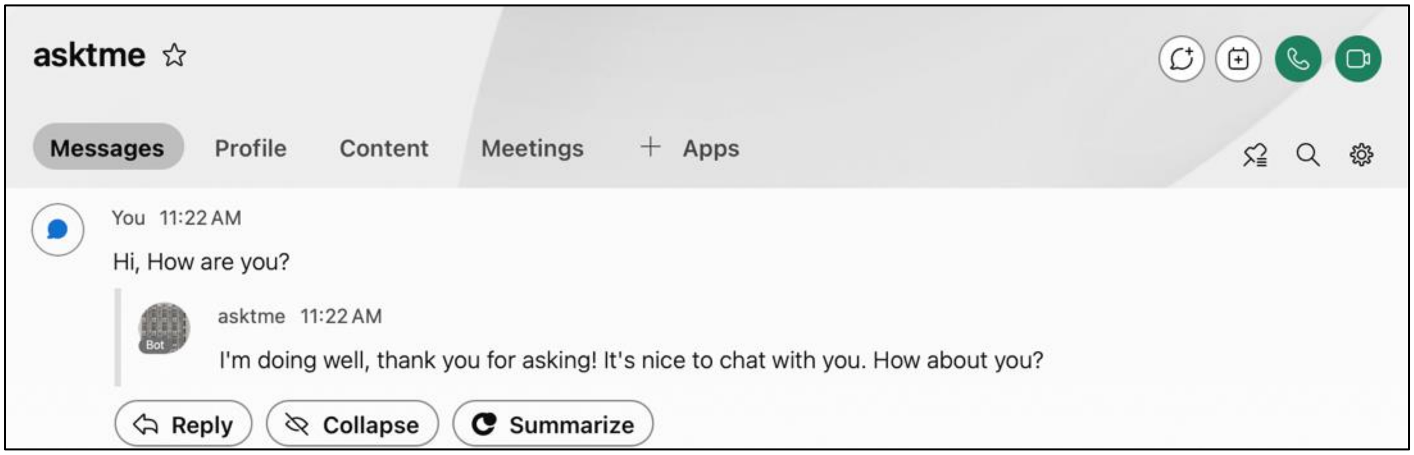
Step 3. Define a router function:

```
def router(info):
    if "general" in info['topic'].lower():
        chain=PromptTemplate.from_template(general_prompt)
    elif "classification" in info['topic'].lower():
        chain=PromptTemplate.from_template(init_prompt)
    else:
        chain=PromptTemplate.from_template(general_prompt)
    return chain
```

Step 4. Loop through to create chain of tasks and get the final response:

```
def generate_loop(query):
    model=LLM()
    rag_model=LLM(use_knowledge_base=True)
    response='classification'
    try:
        cont=True
        while cont:
            if 'general' in response.lower():
                chain=(
                RunnableLambda(router) |
                model() |
                StrOutputParser()
                )
                cont=False
            elif 'knowledge' in response.lower():
                chain=(
                    RunnableLambda(router) |
                    rag_model() |
                    StrOutputParser()
                )
                cont=False
            else:
                chain=(
                    RunnableLambda(router) |
                    model() |
                    StrOutputParser()
                )
            response=chain.invoke({"topic":response,"question":query})
            if response == 'classification':
                pass
        return response
    except:
        logging.error(traceback.format_exc())
        return "An error occurred"
```

The following are some example screenshots of Inferencing with Webex Bot:

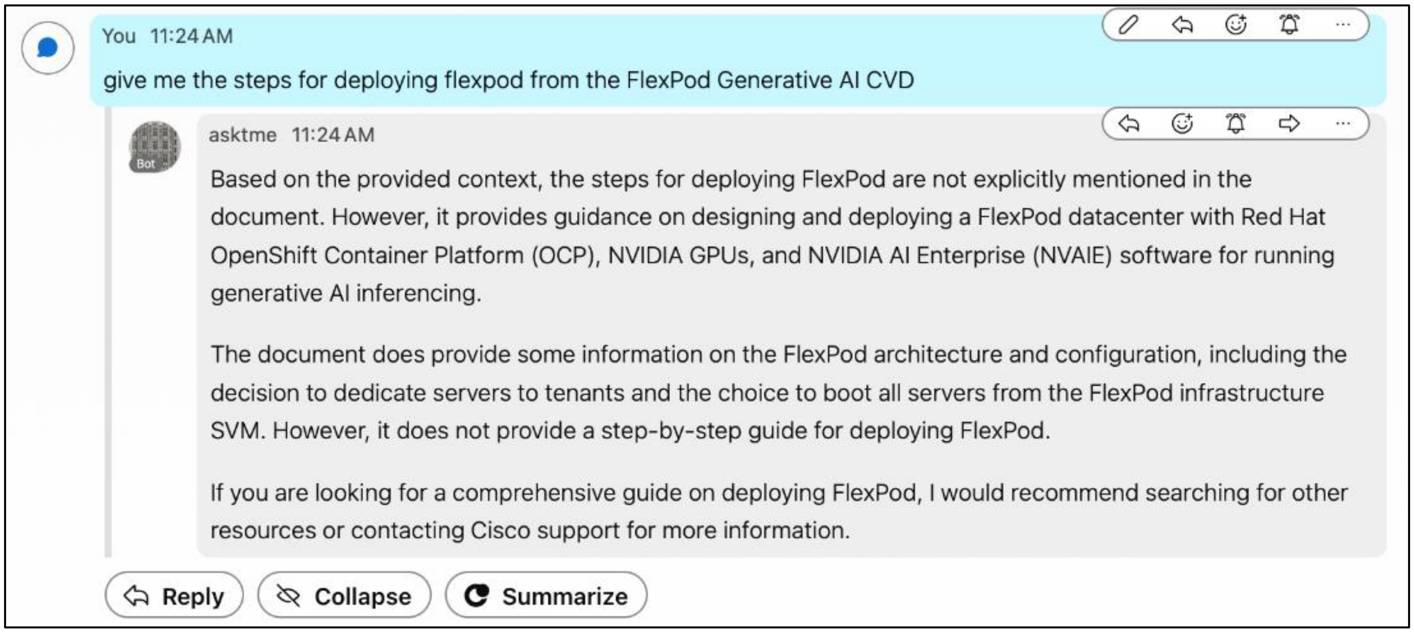


You 11:22 AM

Hi, How are you?

asktme 11:22 AM

I'm doing well, thank you for asking! It's nice to chat with you. How about you?


You 11:24 AM

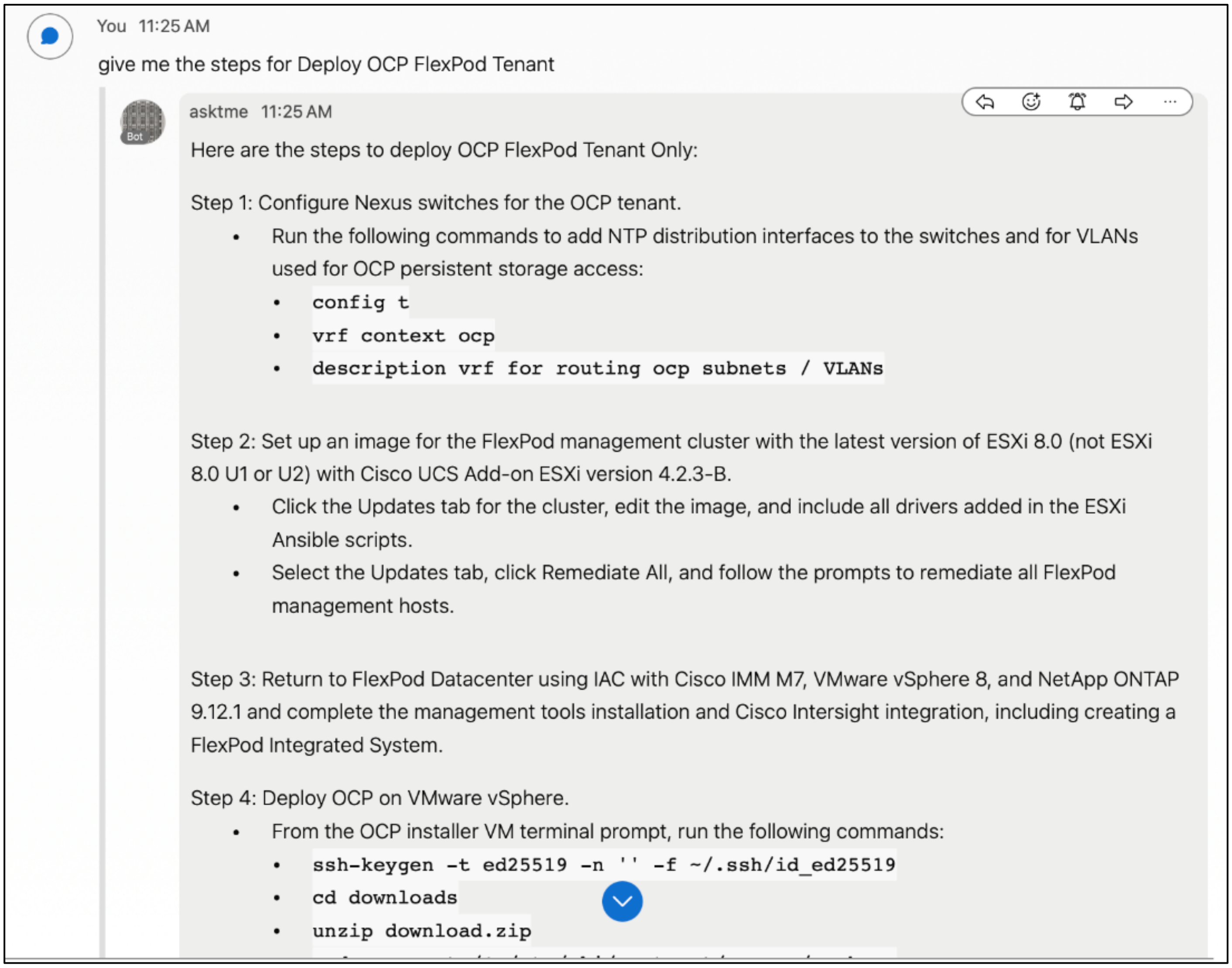
give me the steps for deploying flexpod from the FlexPod Generative AI CVD

asktme 11:24 AM

Based on the provided context, the steps for deploying FlexPod are not explicitly mentioned in the document. However, it provides guidance on designing and deploying a FlexPod datacenter with Red Hat OpenShift Container Platform (OCP), NVIDIA GPUs, and NVIDIA AI Enterprise (NVAIE) software for running generative AI inferencing.

The document does provide some information on the FlexPod architecture and configuration, including the decision to dedicate servers to tenants and the choice to boot all servers from the FlexPod infrastructure SVM. However, it does not provide a step-by-step guide for deploying FlexPod.

If you are looking for a comprehensive guide on deploying FlexPod, I would recommend searching for other resources or contacting Cisco support for more information.



You 11:25 AM

give me the steps for Deploy OCP FlexPod Tenant

asktme 11:25 AM

Here are the steps to deploy OCP FlexPod Tenant Only:

Step 1: Configure Nexus switches for the OCP tenant.

- Run the following commands to add NTP distribution interfaces to the switches and for VLANs used for OCP persistent storage access:
  - `config t`
  - `vrf context ocp`
  - `description vrf for routing ocp subnets / VLANs`

Step 2: Set up an image for the FlexPod management cluster with the latest version of ESXi 8.0 (not ESXi 8.0 U1 or U2) with Cisco UCS Add-on ESXi version 4.2.3-B.

- Click the Updates tab for the cluster, edit the image, and include all drivers added in the ESXi Ansible scripts.
- Select the Updates tab, click Remediate All, and follow the prompts to remediate all FlexPod management hosts.

Step 3: Return to FlexPod Datacenter using IAC with Cisco IMM M7, VMware vSphere 8, and NetApp ONTAP 9.12.1 and complete the management tools installation and Cisco Intersight integration, including creating a FlexPod Integrated System.

Step 4: Deploy OCP on VMware vSphere.

- From the OCP installer VM terminal prompt, run the following commands:
  - `ssh-keygen -t ed25519 -n '' -f ~/.ssh/id_ed25519`
  - `cd downloads`
  - `unzip download.zip`

About the Authors

Paniraja Koppa, Technical Marketing Engineer, Cisco Systems, Inc.

Paniraja Koppa is a member of the Cisco Unified Computing System (Cisco UCS) solutions team. He has over 15 years of experience designing, implementing, and operating solutions in the data center. In his current role, he works on design and development, best practices, optimization, automation and technical content creation of compute and hybrid cloud solutions. He also worked as technical consulting engineer in the data center virtualization space. Paniraja holds a master's degree in computer science. He has presented several papers at international conferences and speaker at events like Cisco Live US and Europe, Open Infrastructure Summit, and other partner events. Paniraja's current focus is on Generative AI solutions.

Mahesh Pabba, Customer Delivery Architect, Cisco Systems, Inc.

Mahesh Pabba is a seasoned technology professional with vast experience in the IT industry. With a strong foundation in enterprise applications, he has successfully navigated various domains, excelling in automation and development using cutting-edge technologies. As a skilled Solution Architect, he has designed and implemented numerous innovative solutions throughout his career. Additionally, he possesses expertise in artificial intelligence, specializing in training, inferencing, and deploying AI solutions on Cisco hardware, encompassing compute, network, and storage environments. His unique blend of technical acumen and solution-oriented approach positions him as a valuable contributor to any AI design initiative.

Keith Brant, Customer Delivery Architect, Cisco Systems, Inc.

Keith has more than 25 years of experience in IT, 15 of which have been at Cisco. He focuses on Compute, Virtualization, Hyperconverged Infrastructure, and AI. Keith earned a bachelor's degree in Computer Management Systems.

Anurag Guda, Technical Marketing Engineer, NVIDIA

Anurag Guda is a technical marketing engineer at NVIDIA, focusing on building AI software solutions on cloud-native technologies. He helps customers to deploy applications on the EGX platform with NVIDIA Certified Systems. He also works closely with leading ISVs, and OEMs partners to bring AI solutions to enterprise. Anurag has an M.Sc. in engineering and technology from Murray State University.

Kamini Singh, Technical Marketing Engineer, Hybrid Cloud Infra & OEM Solutions, NetApp Inc.

Kamini Singh is a Technical Marketing engineer at NetApp. She has more than five years of experience in data center infrastructure solutions. She focuses on FlexPod hybrid cloud infrastructure solution design, implementation, validation, automation, and sales enablement. Kamini holds a bachelor's degree in Electronics and Communication and a master's degree in Communication Systems.

Kavyashree Mahadevaiah, Technical Marketing Engineer, Hybrid Cloud Infra & OEM Solutions, NetApp Inc.

Kavyashree Mahadevaiah is a Technical Marketing Engineer for the FlexPod solutions team and has more than 2 years of experience in Data Center infrastructure solutions which includes On-prem, Hybrid cloud space. She focuses on the design, implementation, validation and automation of converged infrastructure and hybrid cloud infrastructure solutions. Kavyashree holds a Bachelor's degree in Electronics and Communication.

Acknowledgements

For their support and contribution to the design, validation, and creation of this Cisco Validated Design, the authors would like to thank:

- John George, Technical Marketing Engineer, Cisco Systems, Inc.
- Chris O'Brien, Senior Director, Cisco Systems, Inc.
- Dwight Ludington, Customer Delivery Architect, Cisco Systems, Inc.
- Rohit Mittal, Product Manager, Cisco Systems, Inc.
- Urshila Lohani, Product Manager, Cisco Systems, Inc.
- Abhinav Singh, Senior Technical Marketing Engineer, NetApp Inc.
- Sriram Sagi, Principal Product Manager, FlexPod, NetApp Inc.
- Bobby Oommen, Senior Manager, FlexPod Solutions, NetApp Inc.

Feedback

For comments and suggestions about this guide and related guides, join the discussion on Cisco Community at https://cs.co/en-cvds.

CVD Program