VULTR

Deploying the AMD Enterprise AI Blueprint with NetApp FlexCache on Vultr Cloud using AMD Instinct GPUs and AMD AI Microservices (AIMs)

Table of Contents

Introduction

Enterprises are increasingly adopting generative AI applications that interact with internal knowledge bases while maintaining strict governance over sensitive data. Many organizations store critical business documents in on-premises systems and cannot fully migrate these datasets to public cloud environments due to compliance, security, or operational constraints.

Vultr Cloud offers on-demand access to a variety of AMD Instinct GPUs, enabling organizations to deploy high-performance AI and compute workloads on globally available infrastructure. Beyond GPU access, Vultr provides a comprehensive cloud platform with services such as bare metal, Kubernetes, block and object storage, file systems, snapshots, backups, VPC networking, load balancers, and security controls, helping customers build production-ready, scalable cloud environments around AMD-powered acceleration.

This reference architecture demonstrates how to deploy a production-ready Retrieval-Augmented Generation (RAG) platform using AMD Instinct™ GPUs, AMD Inference Microservice (AIMs), Qdrant Vector DB, and NetApp FlexCache hybrid storage on Vultr Cloud. The solution enables GPU-accelerated AI services to retrieve enterprise knowledge while maintaining control over data location and governance.

Key Advantages

This hybrid architecture delivers several advantages for enterprise AI deployments:

- **Data Sovereignty Preservation -** Enterprise datasets remain in on-premises storage while AI services access them through controlled caching mechanisms.
- **Low-Latency Enterprise Data Access** - NetApp FlexCache enables AI workloads to read documents directly from the extended storage namespace without replicating entire datasets to the cloud.
- **Efficient AI Inference Infrastructure** - AMD Instinct™ GPUs combined with AMD Inference Microservice (AIMs) provide optimized runtime environments for large-scale model inference.
- **Independent Component Scaling** - Storage, vector indexing, inference services, and application layers can scale independently to support evolving workload demands.
- **Accelerated Enterprise AI Deployment -** Pre-validated architecture patterns and containerized model services simplify the deployment of production-grade generative AI applications.
- **Governed AI Workloads -** Existing enterprise security, compliance, and data governance policies remain intact while enabling AI systems to interact with internal knowledge sources.

Core Platform Components

The platform brings together GPU-accelerated inference, semantic retrieval, and hybrid data access into a single deployment model. AI services are deployed on AMD Instinct™ GPUs using AMD Inference Microservices (AIMs). These microservices handle model execution and expose APIs for application integration.

Document content and user queries are converted into vector embeddings and stored in Qdrant, enabling efficient semantic search and retrieval. During query processing, relevant context is retrieved from Qdrant and passed to the language model for response generation.

Enterprise data remains on NetApp ONTAP, with FlexCache providing low-latency access to datasets from the cloud environment.

The architecture integrates the following components:

- **AMD Enterprise AI Suite** - Provides the Kubernetes-based AI platform used to deploy and manage GPU-accelerated AI services, including model inference, storage integration, and application workloads.
- **NetApp FlexCache** - Extends enterprise storage into the cloud while maintaining the authoritative data source on-premises.
- **AMD Instinct™ MI325X GPUs** - Provide hardware acceleration for embedding generation and large language model execution.
- **AMD Inference Microservice (AIMS)** - Deploys and manages GPU-accelerated AI model inference through containerized runtime environments.
- **Embedding Service** - Converts document content and user queries into vector representations used for semantic search.
- **Qdrant Vector Database** - Performs semantic similarity search over document embeddings.
- **Retrieval-Augmented Generation (RAG) Pipeline** - Retrieves relevant document context and generates responses grounded in enterprise knowledge.
- **Helm Deployment** - Installs and manages the blueprint application components on the Kubernetes cluster.

By separating storage, vector indexing, AI inference services, and application layers, the architecture enables independent scaling of infrastructure while providing a flexible and scalable foundation for enterprise AI workloads.

AMD Enterprise AI Suite

AMD Enterprise AI Suite provides the Kubernetes-based platform used in this guide to deploy and operate AI workloads on **AMD Instinct GPU infrastructure**. Throughout this guide, *platform* refers to the environment delivered by AMD Enterprise AI Suite.

The platform integrates containerized model services, open-source AI frameworks, storage, networking, and GPU scheduling into a unified environment for running AI workloads. **AMD Solution Blueprints** build on this platform to deliver validated architectures for applications such as **Retrieval-Augmented Generation (RAG),** document intelligence, and conversational AI systems.

Key Features

- Full-stack AI platform for deploying and operating AI workloads on AMD GPU infrastructure
- ROCm-based software ecosystem optimized for AMD Instinct GPU acceleration
- Kubernetes orchestration for scalable deployment and lifecycle management
- **AMD AI Blueprints** providing validated architectures for enterprise AI applications
- Integration with open-source AI frameworks and model ecosystems
- Support for hybrid and multi-cloud AI deployments

Enterprise Benefits

- Accelerates enterprise AI adoption using validated architecture patterns
- Simplifies deployment of scalable AI platforms across infrastructure environments
- Enables consistent operations for AI workloads across teams and projects
- Improves infrastructure utilization through standardized GPU-enabled deployments
- Provides a foundation for production-grade AI services

AMD Inference Microservice (AIMs)

AMD Inference Microservice (AIMs) provides standardized inference microservices for running AI models on **AMD Instinct GPU infrastructure.** It simplifies deployment of large language models, embedding models, and other inference workloads by offering pre-configured containerized runtime environments optimized for the **ROCm software stack.**

AIMs automatically configures runtime parameters based on the selected model and available GPU resources, enabling efficient model execution without extensive manual tuning. It also exposes **OpenAI-compatible APIs,** allowing AI applications, frameworks, and RAG pipelines to integrate with GPU-accelerated inference services using standard interfaces.

Key Features

- Containerized inference services for LLMs, embedding models, and ML workloads
- Hardware-aware runtime optimization for AMD Instinct GPUs
- Automatic models download and caching from model repositories such as Hugging Face
- Multiple deployment profiles optimized for latency or throughput
- OpenAI-compatible APIs for easy integration with AI applications
- Kubernetes-native deployment for scalable inference services

Enterprise Benefits

- Simplifies deployment of generative AI models on AMD GPU infrastructure
- Reduces operational complexity through standardized inference services
- Improves GPU utilization through optimized runtime configurations
- Accelerates development of AI applications through API-driven integration
- Supports scalable AI inference services for enterprise production workloads

AMD AI Solution Blueprints

Solution Blueprints are reference applications built with AIMs and offers an easy way to explore AMD AI Suite in the context of a complete microservice solution. For developers, Solution Blueprints act as starting points and example implementations, making it fast and easy to solve real-world needs with ROCm software.

The industry-standard orchestration system for microservices such as AIMs is Kubernetes. Solution Blueprints are packaged as helm charts, a templating software for Kubernetes manifests. Solution Blueprint charts are off-the-shelf ready to deploy to an AMD Enterprise AI Suite cluster, and templating allows the applications to be customized.

Hybrid Storage Architecture Using NetApp FlexCache

AI workloads require fast access to enterprise datasets, but migrating large volumes of data to the cloud is often impractical. NetApp FlexCache addresses this challenge by extending on-premises ONTAP volumes into cloud environments without requiring full dataset replication.

In this architecture:

- The on-premises NetApp ONTAP system remains the authoritative source of enterprise data.
- A FlexCache volume is deployed in Vultr Cloud and mounted by the AI application host through NFS.

FlexCache dynamically retrieves frequently accessed files from the origin ONTAP system and caches them near the cloud compute infrastructure. This allows GPU-based AI workloads to access enterprise datasets with minimal latency while maintaining centralized storage governance.

Key Characteristics

- Enterprise data remains stored on-premises in NetApp ONTAP
- FlexCache extends the storage namespace into the cloud environment
- AI workloads access enterprise files through standard NFS mounts
- Frequently accessed files are cached near GPU/compute resources
- No bulk data migration or duplicate storage estates are required

Enterprise Benefits

- Low-latency access to enterprise datasets for AI workloads
- Preserves existing governance and compliance policies
- Reduces storage costs by avoiding full dataset replication
- Simplifies hybrid-cloud data access for AI pipelines
- Enables scalable multi-region AI deployments for enterprise AI platforms

Architecture Overview

The architecture combines hybrid storage, application services, vector search, and GPU-accelerated AI inference into a unified enterprise AI platform.

The system is composed of three primary layers:

- Hybrid Data Architecture
- Compute and AI Services Architecture
- Retrieval-Augmented Generation (RAG) Processing Pipeline

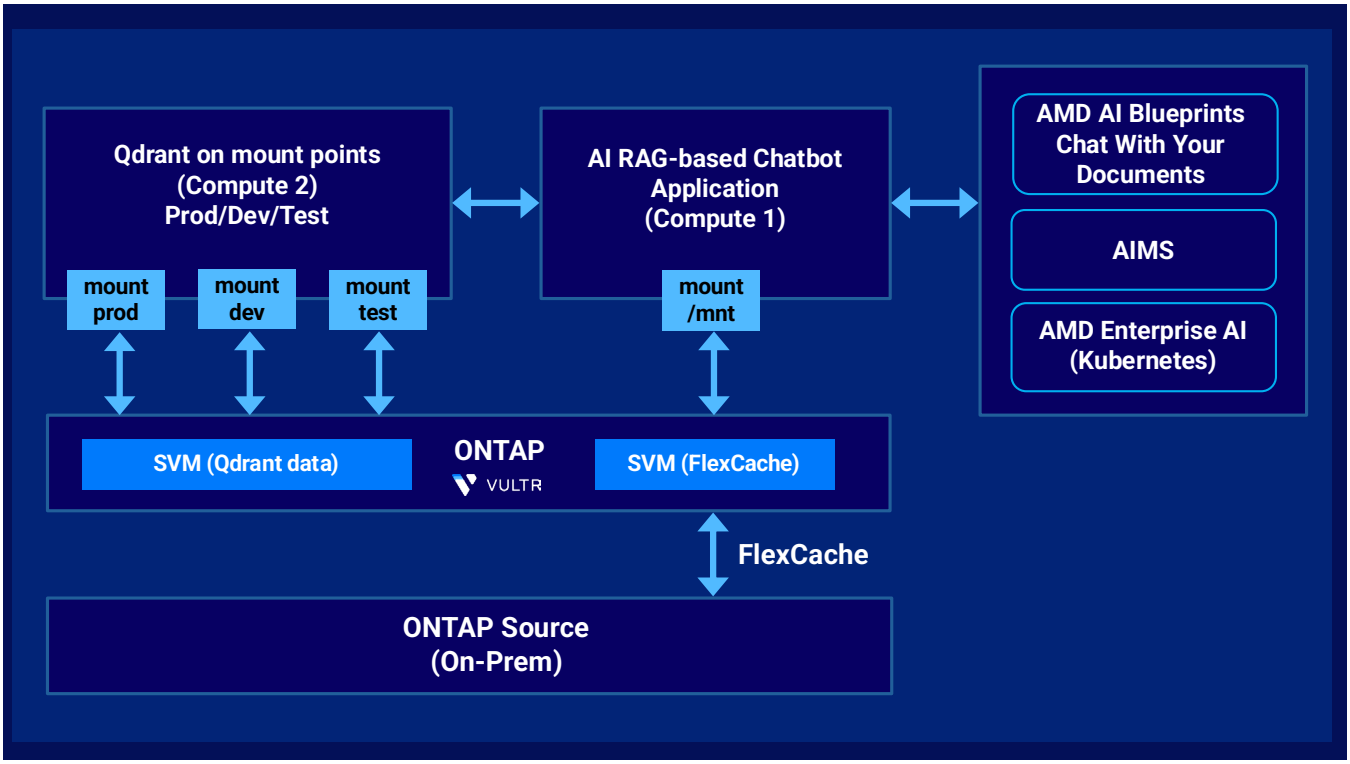

Figure 1 - Chat with your Document architecture with NetApp FlexCache, AMD AIMs and AMD Enterprise AI

Additionally, this could also be deployed such that the FlexCache can be directly mounted on the GPU node as needed as shown below. In production environments, enterprise storage is typically integrated with Kubernetes through persistent storage mechanisms or mounted directly to compute nodes running AI services.

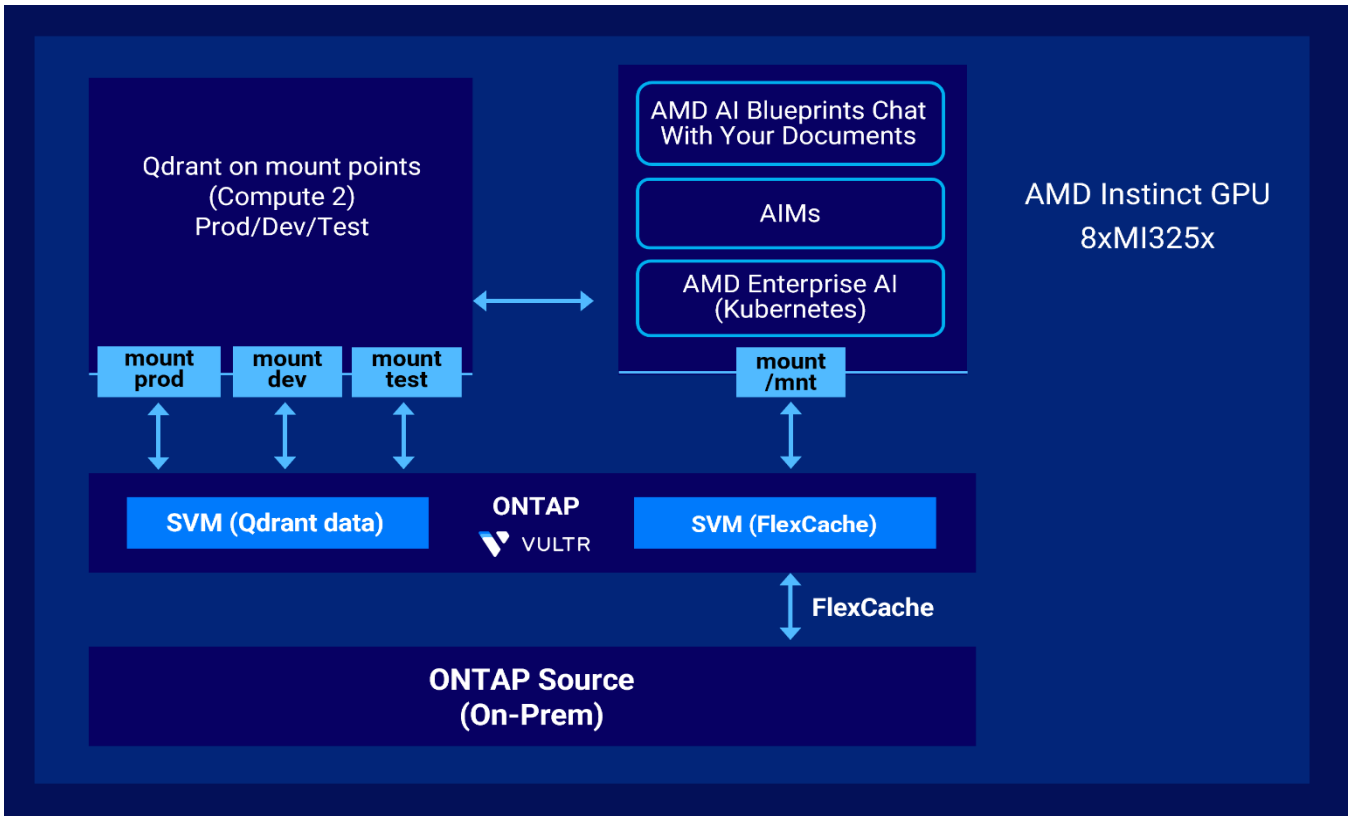

Figure 2 - RAG High-Level Architecture (AMD AIMS +Qdrant + FlexCache)

Hybrid Data Architecture

The **hybrid data architecture** enables AI services running in **Vultr Cloud** to access enterprise datasets stored in the on-premises **NetApp ONTAP** environment. A **FlexCache volume** deployed in Vultr Cloud provides a cached view of the ONTAP storage volumes and is mounted to the AI application host using **NFS**.

Through this mounted file system path, **AI applications and GPU-accelerated services** can access enterprise documents directly from the FlexCache volume, while the **authoritative data remains stored on-premises.** This approach enables **low-latency access to frequently used data** without requiring separate data replication workflows, while preserving a **single authoritative data source.**

Compute and AI Services Architecture

This layer hosts the application, vector database, and GPU-based inference services required to process documents and serve user queries.

Qdrant Vector Database

A dedicated compute instance hosts the Qdrant vector database used for semantic indexing and retrieval.

Independent collections are maintained for Production, Development, and Test environments to support controlled upgrades and safe experimentation.

Configuration

- 2 vCPUs
- 8 GB RAM

Key Functions

- Store vector embeddings generated from enterprise documents
- Perform similarity search operations
- Return the most relevant document chunks during query time

Separating the vector database from the application layer ensures scalability and predictable query performance.

AI Inference Layer

AI inference services run on a GPU server equipped with AMD Instinct MI325X GPUs in Vultr Cloud. Model inference is orchestrated through AMD Inference Microservice (AIMs), which deploys the embedding and language models onto the AMD Instinct GPU infrastructure.

GPU allocation

- 1 GPU - Embedding model (intfloat/multilingual-e5-large-instruct)
- 2 GPUs - Large Language Model served via AIMs (llama-3-3-70b-instruct)

The LLM uses tensor parallelism (TP=2) across two GPUs to support distributed model execution.

This configuration enables efficient inference for large models while maintaining low latency and consistent throughput for RAG-based query workloads.

Retrieval Augmented Generation (RAG) Pipeline

The RAG pipeline generates responses grounded in enterprise documents instead of relying solely on model pre-training.

The pipeline operates in two stages:

- **Offline ingestion stage -** builds a semantic knowledge index from enterprise documents
- **Online inference stage -** retrieves relevant context and generates responses in real time

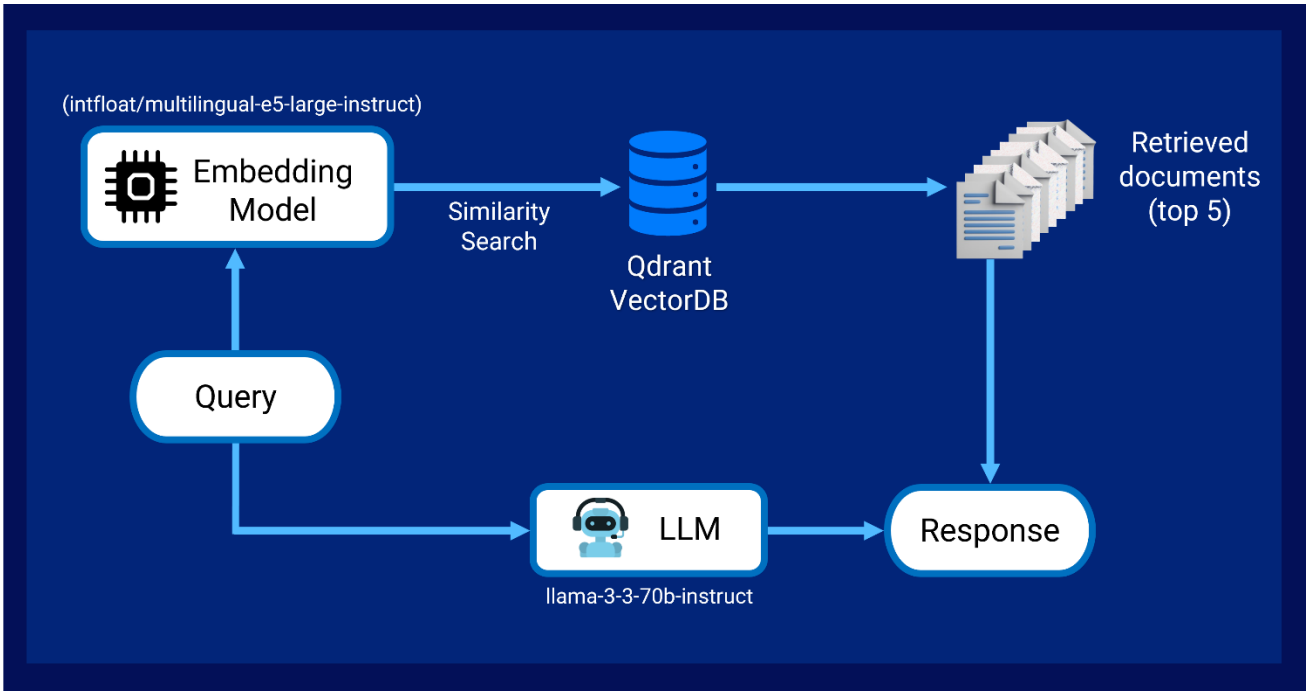

Figure 3 - Offline ingestion and online retrieval flow for the document chat application

Offline Ingestion - Building the Knowledge Index

Before the system can answer questions, enterprise documents must be processed and indexed into a semantic representation stored in the **Qdrant vector database.**

Step 1 - Read

- Documents are accessed directly from the **NFS-mounted NetApp FlexCache volume,** allowing the system to process enterprise data without copying or staging files.

Step 2 - Parse

- Documents are parsed using **LangChain document loaders** to extract text from formats such as PDFs and structured document files.

Step 3 - Chunk

- Extracted text is divided into overlapping chunks to preserve contextual continuity across document sections.

Step 4 - Embed

- Each chunk is processed by the **intfloat/multilingual-e5-large-instruct** embedding model served through **AMD Inference Microservice (AIMs)** on an **AMD Instinct MI325X GPU,** generating semantic vector embeddings.

Step 5 - Index in Qdrant

- Generated embeddings are stored in **Qdrant,** which indexes the vectors for efficient similarity search during query processing.

Online Inference - Answering User Queries

During runtime, user queries are encoded, matched against the semantic index, and answered using GPU-accelerated LLM inference.

Step 1 - Encode Query

- The user query is converted into a vector embedding using the **intfloat/multilingual-e5-large-instruct** model, the same embedding model used during document ingestion.

Step 2 - Retrieve

- The query embedding is sent to **Qdrant**, which performs a similarity search and returns the most relevant document chunks from the indexed dataset.

Step 3 - Generate Response

- The retrieved document chunks and the user query are combined into a prompt and sent to the **llama-3-3-70b-instruct** model served through **AMD Inference Microservice (AIMs)** on **AMD Instinct MI325X GPUs.** The model runs with **tensor parallelism (TP=2)** across two GPUs to perform distributed inference.

The LLM generates a response grounded entirely in the retrieved enterprise document context, which is then returned to the application interface for display to the user.

Deployment Overview

This build guide describes how to deploy the AMD Blueprint "Chat with Your Documents", a Retrieval-Augmented Generation (RAG) application running on the AMD Enterprise AI Suite. The deployment provisions GPU-accelerated inference services using **AMD Instinct™ MI325X GPUs,** integrates semantic retrieval through the **Qdrant vector database,** and connects enterprise documents stored in **NetApp ONTAP FlexCache.**

Application services are deployed on the Kubernetes-based AMD Enterprise AI Suite environment using **Helm,** enabling organizations to operationalize a production-ready document intelligence solution on AMD infrastructure.

Deployment Workflow

The deployment process consists of the following stages:

- Configure networking and **NetApp FlexCache storage access** for enterprise documents.
- Install the **AMD Enterprise AI Suite** with Bloom and enable GPU resources.
- Deploy the **Qdrant vector database** used for storing and retrieving document embeddings.

- Deploy the **"Chat with Your Documents"** blueprint using Helm, including embedding and LLM inference services.
- Validate deployment by confirming document ingestion, semantic retrieval, and response generation.

Prerequisites

Ensure the following infrastructure and tools are available.

Infrastructure Requirements

- An **AMD Enterprise AI Suite cluster** installed with **Bloom** and at least one **AMD MI325X GPU** node.
- **SSH access** to the following systems:
    - Helm deployment node
    - FlexCache mount node
    - Kubernetes control plane

Required Tools

The following tools must be installed on the **Helm deployment node:**

- **kubectl -** Kubernetes cluster management
- **helm -** Deployment of blueprint components
- **git -** Cloning the blueprint repository
- **ssh -** Secure node access
- **rsync -** Synchronization of documents from the FlexCache mount
- If Qdrant is deployed outside Kubernetes, ensure **Docker** is installed on the Qdrant host.

Network and Security Configuration

- Allow **HTTPS ingress (port 443)** for accessing the application interface.
- Ensure the required ports (for example, **6333 for Qdrant**) are accessible from the **Kubernetes cluster** to the **external Qdrant instance.**

Baseline Environment Requirements

This guide assumes the following baseline environment:

- **Source SVMs and volumes** available on the ONTAP cluster (Source & Destination).
- **FlexCache storage** configured in the **Vultr Cloud environment**.
- A GPU node with at least **three AMD Instinct GPUs** and **ROCm installed.**

- Compute hosts are available for running the AI application and the Qdrant vector database.
- A secure **IPSec tunnel (or equivalent)** established between the on-premises ONTAP environment and the Vultr cloud region.
- Administrative access to the **ONTAP clusters and the Vultr console or API.**

Preparing these baseline components ensures that the enterprise data storage, AI compute infrastructure, and networking connectivity are ready before deploying the RAG application stack.

> ⓘ Note on Example IP Addresses
>
> - This guide uses RFC 5737 documentation-reserved IP addresses (e.g., 192.0.2.XX) throughout all examples and command output. These are non-routable addresses designated by IANA specifically for use in documentation.
> - Replace them with your actual infrastructure IPs before deployment.

Deployment Scenario

For this AMD AI Solution Blueprint deployment, following is the architecture used.

- The AMD Enterprise AI Suite runs on a single node with **8× AMD Instinct MI325X GPUs.**
- The **FlexCache mount** point resides on a separate compute instance.
- Enterprise documents are synchronized from FlexCache to the Kubernetes node using **rsync**.
- The synchronized documents are then mounted or copied into the RAG application pod for ingestion.

If the FlexCache mount already contains pre-populated documents, the deployment script (`deploy.sh` used for deploying blueprint) can automatically synchronize and index the documents during application deployment.

Prepare FlexCache Volume access for Data Ingestion

Mount the FlexCache export on the compute node before running the deployment script. Use a stable absolute path and verify that the NFS client tools are installed. The mount must be accessible so that document synchronization can complete during deployment.

Step 1 | Configure volume export/security settings (NFS/SMB)

Export policies define which compute nodes can mount the FlexCache volume via NFS or SMB. Vultr compute instances must be explicitly granted both read-write access to the destination volumes. We will be configuring the following:

- **export-policy create** - Creates an export policy that defines the overall access framework for NFS clients on the SVM.
- **export-policy rule create** - Adds specific access rules (read-only/read-write, security type, client conditions) to the policy so ONTAP knows what permissions to enforce.
- **volume modify -policy <policy-name>** - Applies the export policy to a specific volume, enabling ONTAP to enforce those access rules when the volume is mounted.
- **volume mount -junction-path <path>** - Mounts the volume into the SVM's namespace, making it visible and accessible for NFS clients to mount.

```
vserver export-policy create -vserver dst_svm -policyname <policy-name>
```

```
ONTAPSelectCluster::*> export-policy create -policyname Test_dest_flexpolicy -vserver SVM_Dataflex
```

Validate:

```
export-policy show
```

```
ONTAPSelectCluster::*> export-policy show -policyname Test_dest_flexpolicy
Vserver        Policy Name
-------------- --------------------
SVM_Dataflex   Test_dest_flexpolicy
```

Expected Output:

- Export policy **Test_dest_flexpolicy** created successfully
- Associated with the **Vserver SVM_Dataflex**
- Status indicates the policy is **active** and ready to be configured with rules
- Confirms policy is ready to control NFS/CIFS access for volumes under the Vserver

Add rule:

- After creating the policy, associate the required hosts with the policy and assign the appropriate permission levels.

```
vserver export-policy rule create -vserver dst_svm -policyname <policy-name>-clientmatch
<nfs_node_on_vultr> -rorule any -rwrule any -protocol nfs3 –superuser any
```

```
ONTAPSelectCluster::*> vserver export-policy rule create -policyname Test_dest_flexpolicy -vserver
SVM_Dataflex -clientmatch 192.0.2.7 -rorule any -rwrule any -superuser any
```

Validate:

```
export-policy rule show -policyname <policy-name>
```

```
ONTAPSelectCluster::*> export-policy rule show -policyname Test_dest_flexpolicy

Vserver        Policy Name           Rule Index  Access Protocol  Client Match     RO Rule

-------------- -------------------- ---------- -------------- ---------------- ----------

SVM_Dataflex   Test_dest_flexpolicy 1          any             192.0.2.7        any
```

Expected Output:

- Rule added successfully to **export policy Test_dest_flexpolicy**
- Shows **client match:** 192.0.2.7
- Read-only (**rorule**) and read-write (**rwrule**) permissions set (**any**)
- Protocol applied: **NFSv3**
- Superuser access: **any**
- Confirms the rule is **active** and ready to allow NFS access for the client

Step 2 | Create Destination NetApp FlexCache Volume

A NetApp FlexCache volume stores hot (frequently accessed) data close to users or applications, while the origin volume remains the authoritative source. FlexCache automatically manages cache coherency, metadata synchronization, and write-forward behavior, ensuring consistent access across sites without administrative overhead.

FlexCache volumes do not need to match the size of the origin volume. Because they store only hot data, metadata, and recently accessed blocks, they can be provisioned significantly smaller - typically 5 - 20% of the origin volume's capacity - making them highly storage-efficient.

CLI

Create a NetApp FlexCache volume

The FlexCache volume is created on the destination cluster/SVM and linked to the source volume. Only metadata and frequently accessed data are cached, reducing latency and WAN traffic.

```
volume flexcache create  -vserver <cache_svm>  -volume <cache_volume_name>  -aggr-list <aggregate>
-size 1TB  -origin-volume <origin_vol>  -origin-vserver <origin_svm>
```

```
ONTAPSelectCluster::*> flexcache create -vserver SVM_Dataflex -volume Flex_vol_Test01 -size 7GB -
aggr-list aggr_data -origin-volume  Flex_Vol_S20 -origin-vserver SVM_Flexdata

(volume flexcache create)

[Job 128] Job succeeded: Successful.
```

- volume flexcache create

  Initiates the creation of a FlexCache volume on the destination (cache) cluster.
- -vserver <cache_svm>

  Specifies the Storage Virtual Machine (SVM) where the FlexCache volume will be created.
- -volume <cache_volume>

  Defines the name of the FlexCache volume on the cache SVM.
- -origin-vserver <source_svm>

  Identifies the source SVM that hosts the original (origin) volume.
- -origin-volume <source_volume>

  Specifies the source volume whose data will be cached in the FlexCache volume.
- -size <size>

  Sets the size of the FlexCache volume (used for metadata and cached data blocks).

Validate from Destination cluster

- Confirms the cache is correctly associated with the origin volume.

```
volume flexcache show
```

```
ONTAPSelectCluster::*> flexcache show
  (volume flexcache show)
Vserver Volume      Size       Origin-Vserver Origin-Volume Origin-Cluster
------- ----------- ---------- -------------- ------------- --------------
SVM_Dataflex Flex_vol_Test01 7GB SVM_Flexdata Flex_Vol_S20  sxId0edb1927eae795ba7
SVM_data Flex_Test_DES01 6GB   sx            Flex_Test_S10 sxId0edb1927eae795ba7
SVM_data Flex_Test_DES01 6GB   sx            Flex_Test_S10 sxId0edb1927eae795ba7
2 entries were displayed.
```

Validate from Source origin NetApp ONTAP cluster

- Validate origin and cache relationship health.

```
volume flexcache origin show
```

```
sxId0edb1927eae795ba7::*>  volume flexcache origin show
Origin-Vserver Origin-Volume  Cache-Vserver  Cache-Volume  Cache-Cluster
-------------- -------------- -------------- ------------- --------------
SVM_Flexdata   Flex_Vol_S20   SVM_Dataflex   Flex_vol_Test01 ONTAPSelectCluster
sx             Flex_Test_S10  SVM_data       Flex_Test_DES01 ONTAPSelectCluster
2 entries were displayed.
```

Expected Output:

- FlexCache volume **Flex_vol_Test01** created successfully
- Associated with **Vserver SVM_Dataflex**
- Linked to **aggregate aggr_data** and sized **7GB**
- Origin volume: **Flex_Vol_S20** on Vserver **SVM_Flexdata**
- Caching relationship with origin volume established and ready to serve cached data to clients

Step 3 | Mount the NetApp FlexCache Volume with Junction Path

The volume mount command attaches the volume to the SVM's namespace at a junction path, making it visible and mountable by NFS clients. Without a junction path, the volume exists internally but cannot be accessed over NFS.

```
volume mount -vserver <dest_svm>  -volume <cache_vol1>  -junction-path /Junction-path
```

```
ONTAPSelectCluster::*> vol mount -volume Flex_vol_Test01 -vserver SVM_Dataflex -junction-path
/Flex_vol_Test01
```

Validation

```
Volume show -volume <volumename> -fields junction-path
```

```
ONTAPSelectCluster::*> vol show -volume Flex_vol_Test01 -fields junction-path
vserver      volume          junction-path
------------ --------------- ---------------
SVM_Dataflex Flex_vol_Test01 /Flex_vol_Test01
```

Expected Output:

- Volume **Flex_vol_Test01** mounted successfully
- Mounted on **Vserver SVM_dataflex**
- Junction path set to **/Flex_vol_Test01**
- Volume status is **online** and accessible to clients
- Confirms the volume is ready for read/write operations

Step 4 | Modify the volume with export-policy

The **volume modify** command associates an export policy with the volume, allowing ONTAP to enforce the appropriate access rules when clients attempt to mount it. Without an export policy assigned, NFS access to the volume is not permitted.

Validate policy before modifying

Validate the existing export policy configuration to ensure the correct policy and rules are identified before making any modifications to the NetApp storage settings.

```
ONTAPSelectCluster::*> vol show -volume Flex_vol_Test01 -fields policy
vserver      volume          policy
------------ --------------- -------
SVM_Dataflex Flex_vol_Test01 default
```

Modify the policy

Modify the export policy to update access rules and permissions, ensuring the required clients are granted appropriate access to the storage resources.

```
volume modify -vserver dst_svm –volume <vol> -policy <policy-name>
```

```
ONTAPSelectCluster::*> vol modify -volume Flex_vol_Test01 -policy Test_dest_flexpolicy -vserver
SVM_Dataflex

[Job 133] Job succeeded: volume modify succeeded
```

Validate policy after modifying

Validate the export policy after modification to ensure the updated rules and permissions are correctly applied and functioning as intended.

```
volume show –volume <volume> -fields policy
```

```
ONTAPSelectCluster::*> vol show -volume Flex_vol_Test01 -vserver SVM_Dataflex -fields policy
vserver      volume          policy
------------ --------------- --------------------
SVM_Dataflex Flex_vol_Test01 Test_dest_flexpolicy
```

Expected Output:

- Volume **Flex_vol_Test01** updated successfully
- Associated with **Vserver SVM_dataflex**
- **Export policy** changed/applied to **Test_dest_flexpolicy**
- Confirms the volume is **online** and the new policy is active
- Volume is ready for client access under the updated export policy

Check Export Policy access

The **export-policy check-access** command in NetApp ONTAP is used to verify whether a specific client is allowed to access an NFS export and what level of access it will receive. It simulates an NFS access request and evaluates the export policy rules without mounting the volume.

The command checks the client IP or hostname against the export policy rules, determines whether access is permitted or denied, and reports the effective permissions such as read-only, read-write, superuser access, and protocol version. This is used for troubleshooting NFS permission issues, helping administrators quickly confirm if export policies are correctly configured for a given client.

```
export-policy check-access -vserver SVM_Dataflex -volume Flex_vol_Test01 -client-ip 192.0.2.7 -
authentication-method sys -protocol nfs3 -access-type read-write
```

If access is denied, add **clientmatch** into the root volume export policy as below:

```
ONTAPSelectCluster::*> export-policy rule create -policyname default -vserver SVM_Dataflex -
clientmatch 192.0.2.7 -rorule any -rwrule any -superuser any
```

Validate:

```
ONTAPSelectCluster::*> export-policy check-access -vserver SVM_Dataflex -volume Flex_vol_Test01 -
client-ip 192.0.2.7 -authentication-method sys -protocol nfs3 -access-type read-write
                                Policy    Policy     Rule            Security
Path                Policy      Owner     Owner Type Index Access    Style
------------------- ----------- --------- ---------- ---- ---------- ----------
/                   default     SVM_Dataflex_root volume 1 read      mixed
/Flex_vol_Test01    Test_dest_flexpolicy Flex_vol_Test01 volume 1 read-write mixed
2 entries were displayed.
```

Expected Output:

- Checks access for client IP 192.0.2.7 on volume **Flex_vol_Test01**
- Shows Vserver: **SVM_Dataflex**
- Authentication method: **sys**
- Protocol: **NFSv3**
- Access type tested: **read-write**
- Output indicates whether access is **allowed** or **denied**
- Confirms which **export policy rule** permits or blocks the access

Step 5 | Mount on Vultr Compute

Verification that Vultr compute nodes can access the replicated/cached dataset over NFS. Ensures the end-to-end path from **on-prem → NetApp FlexCache → NetApp ONTAP (Destination) → Vultr compute** is fully operational.

When we mount an NFS export from NetApp ONTAP(Destination) in Vultr:

- We are mounting the **active filesystem** of the destination NetApp FlexCache Volume
- NetApp FlexCache maintains a local cache of hot (frequently accessed) data from the source volume. Clients can perform read and write operations on the NetApp FlexCache volume - reads are served locally from the cache, while all writes are sent directly to the origin (source) volume, which remains the authoritative data source.

CLI (from compute instance on Vultr Cloud)

```
mount -t nfs <ontap_select_ip>:/<junction-path> <compute node mount point>
```

```
root@compute-01:~# sudo mount -t nfs 192.0.2.53:/Flex_vol_Test01 /mnt/Flexcache_Site2
```

Testing Read Access

Once the FlexCache volume is mounted, we can verify that the end user is able to successfully access and read the cached data from the cache volume.

```
ls -l /mnt/ Flexcache_Site2/
```

```
root@compute-01:/mnt/Flexcache_Site2# ls -l
total 19556
-rw-rw-r-- 1 ubuntu ubuntu 4144691 Mar  2 07:18 ai-generated-images-with-stable-cascade-and-vultr-cloud-gpu.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1602580 Mar  2 07:18 extract-tables-from-images-on-vultr-cloud-gpu.pdf
-rw-rw-r-- 1 ubuntu ubuntu  103115 Mar  2 07:18 how-to-establish-a-private-connection-between-vultr-and-gcp-using-direct-connect-via-console-connect.pdf
-rw-rw-r-- 1 ubuntu ubuntu  569706 Mar  2 07:18 platform-billing.pdf
-rw-rw-r-- 1 ubuntu ubuntu  274552 Mar  2 07:18 platform-security-best-practices.pdf
-rw-rw-r-- 1 ubuntu ubuntu 2053065 Mar  2 07:18 products-cloud-storage.pdf
-rw-rw-r-- 1 ubuntu ubuntu 3019509 Mar  2 07:18 products-compute.pdf
-rw-rw-r-- 1 ubuntu ubuntu  554014 Mar  2 07:18 products-kubernetes.pdf
-rw-rw-r-- 1 ubuntu ubuntu  474028 Mar  2 07:18 products-managed-database-kafka.pdf
-rw-rw-r-- 1 ubuntu ubuntu  424846 Mar  2 07:18 products-managed-database-mysql.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1722812 Mar  2 07:18 products-managed-database.pdf
-rw-rw-r-- 1 ubuntu ubuntu  535387 Mar  2 07:18 products-managed-database-postgresql.pdf
-rw-rw-r-- 1 ubuntu ubuntu  357337 Mar  2 07:18 products-managed-database-valkey.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1341174 Mar  2 07:18 products-network.pdf
-rw-rw-r-- 1 ubuntu ubuntu  427824 Mar  2 07:18 products-orchestration.pdf
-rw-rw-r-- 1 ubuntu ubuntu  388628 Mar  2 07:18 products-serverless-inference.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1883586 Mar  2 07:18 solution-brief-qdrant.pdf
root@compute-01:/mnt/Flexcache_Site2#
```

Testing Write Access (Destination NetApp ONTAP)

The following commands demonstrate the testing of FlexCache write functionality. A file named **output.txt** was successfully created on the cache volume, and it is now visible on the source NFS host, confirming that write operations are correctly redirected to the source volume.

```
root@compute-01:/mnt/Flexcache_Site2# ls -l
total 19560
-rw-rw-r-- 1 ubuntu ubuntu  4144691 Mar  2 07:18 ai-generated-images-with-stable-cascade-and-
vultr-cloud-gpu.pdf
-rw-rw-r-- 1 ubuntu ubuntu  1602580 Mar  2 07:18 extract-tables-from-images-on-vultr-cloud-gpu.pdf
-rw-rw-r-- 1 ubuntu ubuntu  103115  Mar  2 07:18 how-to-establish-a-private-connection-between-
vultr-and-gcp-using-direct-connect-via-console-connect.pdf
drwxr-xr-x 2 nobody nogroup    4096 Mar  7 05:50 output
-rw-rw-r-- 1 ubuntu ubuntu   569706 Mar  2 07:18 platform-billing.pdf
-rw-rw-r-- 1 ubuntu ubuntu  274552  Mar  2 07:18 platform-security-best-practices.pdf
-rw-rw-r-- 1 ubuntu ubuntu  2053065 Mar  2 07:18 products-cloud-storage.pdf
-rw-rw-r-- 1 ubuntu ubuntu  3019509 Mar  2 07:18 products-compute.pdf
-rw-rw-r-- 1 ubuntu ubuntu   554014 Mar  2 07:18 products-kubernetes.pdf
-rw-rw-r-- 1 ubuntu ubuntu   474028 Mar  2 07:18 products-managed-database-kafka.pdf
-rw-rw-r-- 1 ubuntu ubuntu   424846 Mar  2 07:18 products-managed-database-mysql.pdf
-rw-rw-r-- 1 ubuntu ubuntu  1722812 Mar  2 07:18 products-managed-database.pdf
-rw-rw-r-- 1 ubuntu ubuntu   535387 Mar  2 07:18 products-managed-database-postgresql.pdf
-rw-rw-r-- 1 ubuntu ubuntu   357337 Mar  2 07:18 products-managed-database-valkey.pdf
-rw-rw-r-- 1 ubuntu ubuntu  1341174 Mar  2 07:18 products-network.pdf
-rw-rw-r-- 1 ubuntu ubuntu   427824 Mar  2 07:18 products-orchestration.pdf
-rw-rw-r-- 1 ubuntu ubuntu   388628 Mar  2 07:18 products-serverless-inference.pdf
-rw-rw-r-- 1 ubuntu ubuntu  1883586 Mar  2 07:18 solution-brief-qdrant.pdf
root@compute-01:/mnt/Flexcache_Site2#
```

```
root@compute-01:/mnt/Flexcache_Site2/output# date > output.txt
root@compute-01:/mnt/Flexcache_Site2/output# cat output.txt
Sat Mar  7 06:11:16 AM UTC 2026
root@compute-01:/mnt/Flexcache_Site2/output#
```

Testing Write Through to Origin (Source NetApp ONTAP)

ONTAP FlexCache uses **write-through semantics,** where all writes are forwarded directly to the origin volume, preserving a single authoritative source of truth. A write or update issued at the **FlexCache mount** is **forwarded to the origin volume.**

```
ubuntu@ip-192.0.2.99:/mnt/Flex_Vol_S20$ ls -l
total 19560
-rw-rw-r-- 1 ubuntu ubuntu 4144691 Mar  2 07:18 ai-generated-images-with-stable-cascade-and-vultr-
cloud-gpu.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1602580 Mar  2 07:18 extract-tables-from-images-on-vultr-cloud-gpu.pdf
-rw-rw-r-- 1 ubuntu ubuntu  103115 Mar  2 07:18 how-to-establish-a-private-connection-between-
vultr-and-gcp-using-direct-connect-via-console-connect.pdf
drwxr-xr-x 2 root   root      4096 Mar  7 05:50 output
-rw-rw-r-- 1 ubuntu ubuntu  569706 Mar  2 07:18 platform-billing.pdf
-rw-rw-r-- 1 ubuntu ubuntu  274552 Mar  2 07:18 platform-security-best-practices.pdf
-rw-rw-r-- 1 ubuntu ubuntu 2053065 Mar  2 07:18 products-cloud-storage.pdf
-rw-rw-r-- 1 ubuntu ubuntu 3019509 Mar  2 07:18 products-compute.pdf
-rw-rw-r-- 1 ubuntu ubuntu  554014 Mar  2 07:18 products-kubernetes.pdf
-rw-rw-r-- 1 ubuntu ubuntu  474028 Mar  2 07:18 products-managed-database-kafka.pdf
-rw-rw-r-- 1 ubuntu ubuntu  424846 Mar  2 07:18 products-managed-database-mysql.pdf
-rw-rw-r-- 1 ubuntu ubuntu  535387 Mar  2 07:18 products-managed-database-postgresql.pdf
-rw-rw-r-- 1 ubuntu ubuntu  357337 Mar  2 07:18 products-managed-database-valkey.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1722812 Mar  2 07:18 products-managed-database.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1341174 Mar  2 07:18 products-network.pdf
-rw-rw-r-- 1 ubuntu ubuntu  427824 Mar  2 07:18 products-orchestration.pdf
-rw-rw-r-- 1 ubuntu ubuntu  388628 Mar  2 07:18 products-serverless-inference.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1883586 Mar  2 07:18 solution-brief-qdrant.pdf
ubuntu@ip-192.0.2.99:/mnt/Flex_Vol_S20$ cd output/
ubuntu@ip-192.0.2.99:/mnt/Flex_Vol_S20/output$ ls -l
total 0
-rw-r--r-- 1 root root 32 Mar  7 06:11 output.txt
ubuntu@ip-192.0.2.99:/mnt/Flex_Vol_S20/output$ cat output.txt
Sat Mar  7 06:11:16 AM UTC 2026
ubuntu@ip-192.0.2.99:/mnt/Flex_Vol_S20/output$
```

Expected Output:

- Mount is successful
- Read and Write operations success

Create SVM, Volume and Mount point For Qdrant VectorDB

This section prepares persistent storage for the Qdrant vector database using NetApp ONTAP. A dedicated Storage Virtual Machine (SVM) is configured along with network access, an NFS-enabled volume, and appropriate export policies. The resulting storage path is then mounted on the Vultr compute instance where the Qdrant service runs, providing reliable and persistent storage for vector indexes and collections.

Step 1 | Create SVM on NetApp ONTAP for Qdrant

A Storage Virtual Machine (SVM) in ONTAP is a logical storage container that provides secure and isolated data access to clients using protocols such as NFS, SMB, and iSCSI. Each SVM independently manages its own volumes, network interfaces (LIFs), export policies, and security configurations, enabling multiple workloads to operate securely on the same storage cluster.

Create the SVM

```
vserver create -vserver vservername -subtype default -rootvolume rootvolumename -rootvolume-
security-style unix -language C.UTF-8 -snapshot-policy default -aggregate aggrname -data-services
data-iscsi,data-nfs,data-cifs,data-flexcache,data-nvme-tcp
```

```
ONTAPSelectCluster::> vserver create -vserver SVM_qdrant -subtype default -rootvolume
SVM_qdrant_root -rootvolume-security-style unix -language C.UTF-8 -snapshot-policy default -
aggregate aggr_data -data-services data-iscsi,data-nfs,data-cifs,data-flexcache,data-nvme-tcp

[Job 436] Job succeeded:

Vserver creation completed.
```

CLI Validation

```
vserver show -vserver *SVM_qdrant*
```

```
ONTAPSelectCluster::> vserver show -vserver *SVM_qdrant*
                                Admin      Operational Root
Vserver     Type    Subtype     State      State       Volume              Aggregate
----------- ------- ---------- ---------- ----------- ---------------     ----------
SVM_qdrant  data    default     running    running     SVM_ qdrant_root    aggr_data
```

Step 2 | Create data LIF on NetApp ONTAP

A Data LIF (Logical Interface) in NetApp ONTAP is a logical network interface configured with an IP address on the data network and associated with a specific node and physical port. It acts as the client-facing endpoint for storage access, enabling protocols such as NFS, SMB, and iSCSI to serve data from volumes within an SVM. Unlike cluster or inter-cluster LIFs, Data LIFs handle application and user traffic and must be explicitly created and assigned the appropriate data service policy. Without a properly configured Data LIF, client systems cannot mount or access storage volumes, even if basic network connectivity to the cluster is available. For availability and load distribution, it is recommended to configure at least one Data LIF per node involved in serving client data.

Check the available ports

Checking the available port before creating a LIF ensures that the selected network interface is active, properly configured, and free for assignment, enabling successful and reliable LIF deployment.

```
net port show
```

```
ONTAPSelectCluster::> net port show
  (network port show)

Node: ONTAPSelectCluster-01

Port    IPspace        Broadcast Domain  Link  MTU   Speed(Mbps)  Health
                                                     Admin/Oper   Status
------- ------------ ----------------- ----- ----- ------------ -------
e0a     Default       Default           up    1500  auto/auto    healthy
e0b     Default       Default           up    1500  auto/auto    healthy
e0c     Default       Default           up    1500  auto/auto    healthy
3 entries were displayed.

ONTAPSelectCluster::>
```

Use **e0c** which is the **default, stable, and recommended data-capable port** available on ONTAP for data traffic in most deployments.

```
ONTAPSelectCluster::> net int create -vserver SVM_qdrant -lif SVM_qdrant_nfs-lif -service-policy
default-data-files -address 192.0.2.54 -netmask 255.255.240.0 -home-node ONTAPSelectCluster-01 -
home-port e0c
(network interface create)
```

CLI Validation | Check LIF state:

Check the LIF state to verify that the logical interface is active and operational, ensuring proper network connectivity for storage access.

```
network interface show
```

```
ONTAPSelectCluster::> network interface show

Vserver           Logical           Status     Network            Current             Current Is
                  Interface         Admin/Oper Address/Mask       Node                Port    Home
---------------- ---------------- ---------- ----------------- ------------------- ------- ----
ONTAPSelectCluster
           ONTAPSelectCluster-01_mgmt1 up/up    192.0.2.61/20 ONTAPSelectCluster-01 e0a     true
           cluster_mgmt                up/up    192.0.2.60/20 ONTAPSelectCluster-01 e0a     true
           ic1                         up/up    192.0.2.51/20 ONTAPSelectCluster-01 e0a     true
SVM_Dataflex
           SVM_Dataflex_data_01        up/up    192.0.2.53/20 ONTAPSelectCluster-01 e0b     true
SVM_data
           SVM_data_cifs               up/up    192.0.2.52/20 ONTAPSelectCluster-01 e0c     true
SVM_qdrant
           SVM_qdrant_nfs-lif          up/up    192.0.2.54/20 ONTAPSelectCluster-01 e0c     true

6 entries were displayed.
```

```
net int show -role data -fields role,address,status-oper,status-admin,lif,home-port,home-node
```

```
ONTAPSelectCluster::> net int show -role data -fields role,address,status-oper,status-
admin,lif,home-port,home-node
  (network interface show)

vserver      lif                   role  address   home-node  home-port  status-oper status-admin
------------ --------------------  ----- --------  ---------  ---------- ----------  ------------
SVM_Dataflex SVM_Dataflex_data_01  data  192.0.2.53  ONTAPSelectCluster-01  e0b       up     up
SVM_data     SVM_data_cifs         data  192.0.2.52  ONTAPSelectCluster-01  e0c       up     up
SVM_qdrant   SVM_qdrant_nfs-lif    data  192.0.2.54  ONTAPSelectCluster-01  e0c       up     up

3 entries were displayed.
```

Expected Output:

- LIF: SVM_qdrant_nfs-lif
- Address: 192.0.2.54
- Home Node: ONTAPSelectCluster-01
- Current Node: ONTAPSelectCluster-01 (unless it has failed over)
- Home Port: e0c
- Operational Status: up
- Administrative Status: up

Enable NFS on ONTAP

Enable the NFS service so that external clients can access volumes hosted within the SVM.

```
vserver nfs create -vserver <vservername> -v3 enabled -v4.0 enabled
```

CLI

```
vserver nfs create -vserver SVM_qdrant -v3 enabled -v4.0 enabled
```

```
ONTAPSelectCluster::> vserver nfs create -vserver SVM_qdrant -v3 enabled -v4.0 enabled
ONTAPSelectCluster::> nfs show -vserver *SVM_qdrant*
Vserver: SVM_qdrant
        General Access:  true
                    v3:  enabled
                  v4.0:  enabled
                   4.1:  enabled
                   UDP:  enabled
                   TCP:  enabled
                  RDMA:  enabled
  Default Windows User:  -
 Default Windows Group:  -
```

Verify that the NFS server is configured and enabled on the SVM

```
nfs show
```

```
ONTAPSelectCluster::> nfs show -vserver *SVM_qdrant*

Vserver: SVM_qdrant

         General Access:  true
                     v3:  enabled
                   v4.0:  enabled
                    4.1:  enabled
                    UDP:  enabled
                    TCP:  enabled
                   RDMA:  enabled
   Default Windows User:  -
  Default Windows Group:  -
```

Step 3 | Create a volume for Qdrant database

A NetApp volume in ONTAP is a logical storage container created within a Storage Virtual Machine (SVM) that serves as the primary location where application and user data is stored. It is provisioned from an aggregate and presented to clients through supported protocols such as NFS, SMB, or iSCSI. The volume defines the namespace (junction path), capacity, security style, and export policy that control how data is accessed. Without creating and properly configuring a volume, no client system can store or retrieve data from the SVM, even if the network and Data LIF are correctly configured.

```
vol create -vserver vservername -volume volumename -state online -policy default -aggregate
aggrname -size volumesize
```

```
ONTAPSelectCluster::> vol create -vserver SVM_qdrant -volume qdrant_prod -state online -policy
default -aggregate aggr_data -size 100GB
[Job 437] Job succeeded: Successful
```

- volume create

  Initiates the creation of a Qdrant DB volume on the destination cluster.

- -vserver <SVM_Name>

  Specifies the Storage Virtual Machine (SVM) where the Qdrant DB volume will be created.

- -volume <volume_Name>

  Defines the name of the Qdrant DB volume on the SVM.

- -aggr <Aggr_Name>

  Defines the name of the aggr.

- -size <size>

  Sets the size of the Qdrant DB volume.

CLI Validation

```
Vol show -volume volumename -vserver vservername
```

```
ONTAPSelectCluster::> vol show -volume *qdrant_prod* -vserver SVM_qdrant
Vserver     Volume       Aggregate  State   Type  Size   Available  Used%
----------- ------------ ---------- ------- ----  ------ ---------- -----
SVM_qdrant  qdrant_prod  aggr_data  online  RW    100GB  95.00GB    0%
```

Expected Output:

- The volume **qdrant_prod** has been successfully created with a size of 100GB on the aggregate **aggr_data.**
- It is mounted on the Vserver **SVM_qdrant.**
- The volume status is **online** and accessible to clients.
- This confirms that the volume is ready for read/write operations.

Step 4 | Mount the NetApp Qdrant database Volume with Junction Path

The volume mount command attaches the volume to the SVM's namespace at a junction path, making it visible and mountable by NFS clients. Without a junction path, the volume exists internally but cannot be accessed over NFS.

Check the Current Status

Check the current status to verify whether the volume or junction path is already mounted or in use before proceeding with the mount operation.

```
Vol show -vserver vserver name -volume volumename -fields junction-path
```

```
ONTAPSelectCluster::> vol show -vserver  SVM_qdrant -volume qdrant_prod -fields junction-path
vserver    volume      junction-path
---------- ----------- -------------
SVM_qdrant qdrant_prod -
```

Mount the volume

Mount the volume to make the storage accessible on the host system, allowing applications and users to read from and write data to the mounted directory.

```
volume mount -vserver <dest_svm>  -volume <cache_vol1>  -junction-path /Junction-path
```

```
ONTAPSelectCluster::> vol mount -volume qdrant_prod -junction-path /qdrant_prod -vserver
SVM_qdrant
```

Validation

```
Volume show -volume <volumename> -fields junction-path
```

```
ONTAPSelectCluster::> vol show -vserver  SVM_qdrant -volume qdrant_prod -fields junction-   path
vserver    volume      junction-path
---------- ----------- -------------
SVM_qdrant qdrant_prod /qdrant_prod
```

Expected Output:

- Volume **qdrant_prod** mounted successfully
- Mounted on **Vserver SVM_qdrant**
- Junction path set to **/qdrant_prod**
- Volume status is **online** and accessible to clients
- Confirms the volume is ready for read/write operations

Configure volume export/security settings (NFS/SMB)

Export policies define which compute nodes can mount the FlexCache volume via NFS or SMB. Vultr compute instances must be explicitly granted both read-write access to the destination volumes. We will be configuring the following:

- **export-policy create** - Creates an export policy that defines the overall access framework for NFS clients on the SVM.
- **export-policy rule create** - Adds specific access rules (read-only/read-write, security type, client conditions) to the policy so ONTAP knows what permissions to enforce.
- **volume modify -policy <policy-name>** - Applies the export policy to a specific volume, enabling ONTAP to enforce those access rules when the volume is mounted.
- **volume mount -junction-path <path>** - Mounts the volume into the SVM's namespace, making it visible and accessible for NFS clients to mount.

```
vserver export-policy create -vserver dst_svm -policyname <policy-name>
```

```
ONTAPSelectCluster::> export-policy create -policyname Policy_qdrant -vserver SVM_qdrant
```

Validate

```
export-policy show -policyname Policy_qdrant
```

```
ONTAPSelectCluster::> export-policy show -policyname Policy_qdrant
Vserver          Policy Name
---------------  -------------------
SVM_qdrant       Policy_qdrant
```

Expected Output:

- Export policy **Policy_qdrant** created successfully
- Associated with the **Vserver SVM_qdrant**
- Status indicates the policy is **active** and ready to be configured with rules
- Confirms policy is ready to control NFS/CIFS access for volumes under the Vserver

Add rule

After creating the policy, associate the required hosts with the policy and assign the appropriate permission levels.

```
vserver export-policy rule create -vserver dst_svm -policyname <policy-name>-clientmatch
<nfs_node_on_vultr> -rorule any -rwrule any -protocol nfs3 –superuser any
```

```
ONTAPSelectCluster::> export-policy rule create -policyname Policy_qdrant -clientmatch 192.0.2.11
-rorule any -rwrule any -vserver SVM_qdrant -protocol nfs3 -superuser any
```

Validate

```
export-policy rule show -policyname Policy_qdrant -clientmatch 192.0.2.11
```

```
ONTAPSelectCluster::> export-policy rule show -policyname Policy_qdrant -clientmatch 192.0.2.11

Vserver     Policy Name    Rule Index  Access Protocol  Client Match  RO Rule
----------- -------------- ---------- --------------- ------------ -------
SVM_qdrant  Policy_qdrant 1           nfs3            192.0.2.11     any
```

```
export-policy rule show -policyname Policy_qdrant -fields rorule,rwrule
```

```
ONTAPSelectCluster::> export-policy rule show -policyname Policy_qdrant -fields rorule,rwrule

vserver     policyname    ruleindex  rorule  rwrule
----------- ------------ ---------- ------- -------
SVM_qdrant  Policy_qdrant 1          any     any
```

Expected Output:

- Rule added successfully to export policy **Policy_qdrant**
- Shows **client match:** 192.0.2.11
- Read-only (**rorule**) and read-write (**rwrule**) permissions set (**any)**
- Protocol applied: **NFSv3**
- Superuser access: **any**
- Confirms the rule is **active** and ready to allow NFS access for the client

Modify the volume with export-policy

The 'volume modify' command assigns an export policy to the volume, enabling ONTAP to apply the correct access rules when clients attempt to mount it. Without an export policy bound to the volume, ONTAP will not allow any NFS access.

Validate policy before modifying

Validate the existing export policy configuration to ensure the correct policy and rules are identified before making any modifications to the NetApp storage settings.

```
ONTAPSelectCluster::> vol show qdrant_prod -fields policy
vserver    volume      policy
---------- ----------- -------
SVM_qdrant qdrant_prod default
```

Modify the policy

Modify the export policy to update access rules and permissions, ensuring the required clients are granted appropriate access to the storage resources.

```
volume modify -vserver dst_svm –volume <vol> -policy <policy-name>
```

```
ONTAPSelectCluster::> vol modify -volume qdrant_prod -vserver SVM_qdrant -policy Policy_qdrant
Volume modify successful on volume qdrant_prod of Vserver SVM_qdrant.
```

Validate policy after modifying

Validate the export policy after modification to ensure the updated rules and permissions are correctly applied and functioning as intended.

```
volume show –volume <volume> -fields policy
```

```
ONTAPSelectCluster::> vol show -volume qdrant_prod -fields policy
vserver    volume      policy
---------- ----------- -------------
SVM_qdrant qdrant_prod Policy_qdrant
```

Expected Output:

- Volume **qdrant_prod** updated successfully
- Associated with **Vserver SVM_qdrant**
- **Export policy** changed/applied to **Policy_qdrant**
- Confirms the volume is **online** and the new policy is active
- Volume is ready for client access under the updated export policy

Check Export Policy access

The **export-policy check-access** command in NetApp ONTAP is used to **verify whether a specific client is allowed to access an NFS export and what level of access it will receive**. It simulates an NFS access request and evaluates the export policy rules without mounting the volume. The command checks the client IP or hostname against the export policy rules, determines whether access is permitted or denied, and reports the effective permissions such as read-only, read-write, superuser access, and protocol version. This is mainly used for **troubleshooting NFS permission issues,** helping administrators quickly confirm if export policies are correctly configured for a given client.

```
export-policy check-access -vserver SVM_qdrant -volume qdrant_prod -client-ip 192.0.2.11 -
authentication-method sys -protocol nfs3 -access-type read-write
```

Validate

```
ONTAPSelectCluster::> export-policy check-access -vserver SVM_qdrant -volume qdrant_prod -client-
ip 192.0.2.11 -authentication-method sys -protocol nfs3 -access-type read-write
                                Policy    Policy     Rule            Security
Path                Policy      Owner     Owner Type Index Access    Style
------------------- ----------- --------- ---------- ---- ---------- ----------
/                   default     SVM_qdrant_root volume  0 denied     unix
```

If access is denied, add **clientmatch** into the root volume export policy as below:

```
ONTAPSelectCluster::> export-policy rule create -policyname default -vserver SVM_qdrant -
clientmatch 192.0.2.11 -rorule any -rwrule any -superuser any
```

Validate

```
ONTAPSelectCluster::> export-policy check-access -vserver SVM_qdrant -volume qdrant_prod -client-
ip 192.0.2.11 -authentication-method sys -protocol nfs3 -access-type read-write

                                Policy    Policy     Rule            Security
Path                Policy      Owner     Owner Type Index Access    Style
------------------- ---------- --------- ---------- ---- ---------- ----------
/                   default    SVM_qdrant_root volume  1 read       unix
/qdrant_prod        Policy_qdrant qdrant_prod volume   1 read-write mixed
2 entries were displayed.
```

Expected Output:

- Checks access for client IP 192.0.2.11 on volume **qdrant_prod**
- Shows **Vserver: SVM_qdrant**
- Authentication method: **sys**
- Protocol: **NFSv3**
- Access type tested: **read-write**
- Output indicates whether access is **allowed** or **denied**
- Confirms which **export policy rule** permits or blocks the access

Step 5 | Mount on Vultr Compute

Verify that Vultr compute nodes can access the provisioned NetApp storage FlexVolume using NFS ensures the end-to-end path - from **NetApp ONTAP(Vultr) → Vultr compute** - is fully operational.

When a volume is provisioned and mounted via FlexVolume:

- The volume is dynamically attached and mounted to the compute node using the NFS
- The volume is mounted over NFS from NetApp ONTAP, providing persistent storage that is directly accessible to applications running on the compute node
- Applications can perform read and write operations on the mounted volume, with data managed directly by the underlying ONTAP storage system

CLI (from compute instance on Vultr Cloud)

```
mount -t nfs <ontap_select_ip>:/<junction-path> <compute node mount point>
```

```
root@qdrant-compute:~# mount -t nfs 192.0.2.54:/qdrant_prod /mnt/qdrant_prod
```

```
ls /mnt/qdrant_prod
```

```
root@qdrant-compute:~# cd /mnt/qdrant_prod
root@qdrant-compute:/mnt/qdrant_prod# ls
root@qdrant-compute:/mnt/qdrant_prod#
```

Deploy Qdrant Vector DB on prod mount

The **Qdrant** is deployed using **Docker** by installing Docker on the host system, validating network and storage readiness, and running the official Qdrant container image. This enables a consistent and portable deployment model suitable for production environments.

The deployment includes exposing required service ports and mounting the production filesystem (qdrant_prod) to ensure persistent storage for vector data and collections. This approach simplifies management while maintaining an isolated and scalable runtime environment for vector search workloads.

Step 1 | Pre validate docker and qdrant volume mount

Verify that Docker is installed and properly configured on the host system. This validation step ensures that the Docker engine is available, the service (daemon) is running, and the system is ready to support containerized workloads.

Pre-validation typically includes confirming:

- Installed Docker version
- Ensuring that required Qdrant storage is mounted.

Check whether docker is available on the host system

```
root@qdrant-compute:~# docker ps
Command 'docker' not found, but can be installed with:
snap install docker         # version 28.4.0, or
apt  install docker.io      # version 28.2.2-0ubuntu1~24.04.1
apt  install podman-docker  # version 4.9.3+ds1-1ubuntu0.2
See 'snap info docker' for additional versions.
```

If Docker is not already installed or properly configured, install and validate it before proceeding. Otherwise, this step can be skipped.

Installing the docker

```
root@qdrant-compute:~# apt install docker.io -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-buildx docker-compose-v2
docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 10 not upgraded.
```

CLI Validation

Validate the Docker version and status after installation.

```
root@qdrant-compute:~# docker --version
Docker version 28.2.2, build 28.2.2-0ubuntu1~24.04.1
```

Expected Output:

- Docker version <version>, build <build-id>

Step 2 | Verify storage Mount points

```
root@qdrant-compute:~# df -h
Filesystem                Size  Used Avail Use% Mounted on
tmpfs                     794M  1.3M  793M   1% /run
efivarfs                  256K   25K  227K  10% /sys/firmware/efi/efivars
/dev/vda2                  47G  8.9G   36G  20% /
tmpfs                     3.9G     0  3.9G   0% /dev/shm
tmpfs                     5.0M     0  5.0M   0% /run/lock
/dev/vda1                 511M  6.2M  505M   2% /boot/efi
tmpfs                     794M   12K  794M   1% /run/user/0
192.0.2.54:/qdrant_prod   95G  2.1M   95G   1% /mnt/qdrant_prod
```

Expected Output:

- Filesystem 192.0.2.54:**/qdrant_prod** is mounted on **/mnt/qdrant_prod**

Step 3 | Create a storage directory

Create a storage sub directory inside the **qdrant_prod** mount point.

```
root@qdrant-compute:~# mkdir -p /mnt/qdrant_prod/storage
```

CLI Validation

```
root@qdrant-compute:~# ls /mnt/qdrant_prod
storage
```

Step 4 | Deploy Qdrant Container

Run Qdrant using the official Docker image without specifying a tag. By default, Docker pulls the latest available version.

```
root@qdrant-compute:~# docker run -d   --name qdrant   -p 6333:6333   -p 6334:6334   -e
QDRANT__STORAGE__ON_DISK_PAYLOAD=true   -v /mnt/qdrant_prod/storage:/qdrant/storage  qdrant/qdrant
98f84d8bfbdcd9659c49b6a3f074e327d7bad20ae61a9aaeadcfa6277494a05b
```

Container Runtime Parameters

- **-d** - Runs the container in Detached Mode
- **--name** - Assigns the container a readable name
- **-p** - Map a port from the host machine to a port inside the container
- **-e** - Sets an environment variable inside the container
- **-v** - Creates a volume mount that maps the host directory

CLI Validation

```
root@qdrant-compute:~# docker ps
CONTAINER ID   IMAGE          COMMAND           CREATED        STATUS        PORTS       NAMES
98f84d8bfbdc   qdrant/qdrant   "./entrypoint.sh"  7 seconds ago   Up 7 seconds   0.0.0.0:6333-
6334->6333-6334/tcp, [::]:6333-6334->6333-6334/tcp   qdrant
```

Expected Output:

- Image Name = Qdrant
- STATUS = up
- PORTS= 6333-6334->6333-6334/tcp

Deploy a DEV/TEST FlexClone volume for Qdrant database

Once data ingestion to /mnt/qdrant_prod is complete, FlexClone volumes can be created for the DEV and TEST environments. These provide space-efficient, writable copies of the production dataset, allowing development and testing activities to proceed without affecting production.

A FlexClone volume in NetApp ONTAP is created almost instantly from an existing volume or snapshot without duplicating the underlying data, which minimizes additional storage usage, because a FlexClone is a point-in-time copy and does not automatically re-sync with the source, this step should be performed only after the production data load is finalized. This ensures the DEV and TEST environments are based on the intended production baseline for validation and testing.

Create a DEV FlexClone volume for Qdrant database

Step 1 | Create a DEV flexclone volume

Create a development FlexClone to provision a writable copy of the dataset for testing or development activities without impacting the source data.

```
ONTAPSelectCluster::> vol clone create -vserver SVM_qdrant -flexclone qdrant_dev -type RW -parent-vserver SVM_qdrant -parent-volume  qdrant_prod

[Job 494] Job succeeded: Successful

ONTAPSelectCluster::>
```

- Flexclone volume create

  Initiates the creation of a qdrant DB dev flexclone volume on the destination cluster.

- vserver <SVM_Name>

  Specifies the Storage Virtual Machine (SVM) where the qdrant DB dev volume will be created.

- parent-vserver <SVM_Name>

  Indicates the Storage Virtual Machine (SVM) that serves as the source for creating the Qdrant database.

- parent-volume <volume_Name>

  Specifies the name of the parent volume on the SVM from which the Qdrant database volume will be derived.

Step 2 | Check the volume clone status

Check the clone status to verify that the FlexClone volume has been successfully created and is available for use.

```
ONTAPSelectCluster::> volume clone show

                     Parent  Parent        Parent
Vserver FlexClone    Vserver Volume        Snapshot            State     Type

------- ------------ ------- ------------ -------------------- --------- ----

SVM_qdrant qdrant_dev SVM_qdrant qdrant_prod clone_qdrant_dev.2026-03-09_044951.0 online RW

ONTAPSelectCluster::>
```

Expected Output:

- Flex clone volume **qdrant_dev** created successfully
- Mounted on **Vserver SVM_qdrant**
- Volume status is **online** and accessible to clients
- Confirm the snapshot **clone_qdrant_dev.2026-03-09_044951.0** created successfully
- Confirms the volume is ready for read/write operations

Step 3 | Check volume clone snapshot status

Check the cloned snapshot status to confirm that the snapshot-based clone has been successfully created and is available for use.

```
ONTAPSelectCluster::> snap list qdrant_prod
                                                                  ---Blocks---
Vserver  Volume   Snapshot                                  Size Total% Used%
-------- -------- ------------------------------------ -------- ------ -----
SVM_qdrant qdrant_prod
                  weekly.2026-03-01_0015                 1.40MB     0%    2%
                  daily.2026-03-08_0010                   160KB     0%    0%
                  weekly.2026-03-08_0015                41.17MB     0%   41%
                  hourly.2026-03-08_2305                  164KB     0%    0%
                  hourly.2026-03-09_0005                  160KB     0%    0%
                  daily.2026-03-09_0010                   164KB     0%    0%
                  hourly.2026-03-09_0105                  164KB     0%    0%
                  hourly.2026-03-09_0205                  156KB     0%    0%
                  hourly.2026-03-09_0305                  164KB     0%    0%
                  hourly.2026-03-09_0405                  164KB     0%    0%
                  clone_qdrant_dev.2026-03-09_044951.0    144KB     0%    0%
11 entries were displayed.
```

Step 4 | Split the cloned volume

Split the clone volume to convert the FlexClone into a fully independent volume by separating it from its parent dataset.

```
ONTAPSelectCluster::> volume clone split start -vserver SVM_qdrant -flexclone qdrant_dev

Warning: Are you sure you want to split clone volume qdrant_dev in Vserver SVM_qdrant ? {y|n}: y
[Job 495] Job is queued: Split qdrant_dev.

ONTAPSelectCluster::>
```

Step 5 | Check the clone status post-split

Check the clone status after the split to confirm that the FlexClone volume has successfully completed the split process and is now fully independent from the parent volume.

```
ONTAPSelectCluster::> vol clone split show
This table is currently empty.
ONTAPSelectCluster::>
```

```
ONTAPSelectCluster::> vol clone show
This table is currently empty.
ONTAPSelectCluster::>
```

Step 6 | Mount the NetApp Qdrant database Volume with Junction Path

The volume mount command attaches the volume to the SVM's namespace at a junction path, making it visible and mountable by NFS clients. Without a junction path, the volume exists internally but cannot be accessed over NFS.

```
Vol show -vserver <vservername> -volume <volumename> -fields junction-path
vol show -vserver  SVM_qdrant -volume qdrant_dev -fields junction-path
```

Check the Current Status

Check the current status to verify whether the volume or junction path is already mounted or in use before proceeding with the mount operation.

```
ONTAPSelectCluster::> vol show -vserver  SVM_qdrant -volume qdrant_dev -fields junction-path
vserver    volume     junction-path
---------- ---------- -------------
SVM_qdrant qdrant_dev -
ONTAPSelectCluster::>
```

Mount the volume

Mount the volume to make the storage accessible on the host system, allowing applications and users to read from and write data to the mounted directory.

```
volume mount -vserver <dest_svm>  -volume <cache_vol1>  -junction-path /Junction-path
```

```
ONTAPSelectCluster::> vol mount -volume qdrant_dev -junction-path /qdrant_dev -vserver SVM_qdrant
```

Validation

```
Volume show -volume <volumename> -fields junction-path
```

```
ONTAPSelectCluster::> vol show -vserver  SVM_qdrant -volume qdrant_dev -fields junction-path
vserver    volume     junction-path
---------- ---------- -------------
SVM_qdrant qdrant_dev /qdrant_dev
ONTAPSelectCluster::>
```

Expected Output:

- Volume **qdrant_dev** mounted successfully
- Mounted on **Vserver SVM_qdrant**
- Junction path set to **/qdrant_dev**
- Volume status is **online** and accessible to clients
- Confirms the volume is ready for read/write operations

Step 7 | Check the status of the export-policy

Check the status of the export policy to verify that the appropriate access rules are applied.

Note: cloned volume inherits the export policy from its parent by default.

Check the Export-policy status

```
ONTAPSelectCluster::> vol show qdrant_dev -fields policy
vserver    volume     policy
---------- ---------- -------------
SVM_qdrant qdrant_dev Policy_qdrant
ONTAPSelectCluster::>
```

Check the Export-policy rule status

```
ONTAPSelectCluster::> export-policy rule show -vserver SVM_qdrant -policyname Policy_qdrant
             Policy          Rule    Access   Client                RO
Vserver      Name            Index   Protocol Match                 Rule
------------ --------------- ------  -------- --------------------- ---------
SVM_qdrant   Policy_qdrant   1       nfs3     192.0.2.11           any

ONTAPSelectCluster::>
```

Step 8 | Mount on Vultr Compute

Verification that Vultr compute nodes can access the cloned dataset over NFS ensures the end-to-end path - from **NetApp ONTAP(Vultr) → Vultr compute** - is fully operational.

When we mount an NFS export from NetApp ONTAP in Vultr:

- We are mounting the active filesystem of the NetApp FlexClone volume.
- NetApp FlexClone creates a writable clone of the parent volume or snapshot, enabling independent read and write operations without affecting the source data. The clone shares underlying storage blocks with the parent volume, providing space efficiency while maintaining data consistency.

CLI (from compute instance on Vultr Cloud)

```
mount -t nfs <ontap_select_ip>:/<junction-path> <compute node mount point>
```

```
root@qdrant-compute:~# mount -t nfs 192.0.2.54:/qdrant_dev /mnt/qdrant_dev
```

```
ls /mnt/qdrant_dev
```

```
root@qdrant-compute:/mnt/qdrant_dev# cd /mnt/qdrant_dev
root@qdrant-compute:/mnt/qdrant_dev#
root@qdrant-compute:/mnt/qdrant_dev#
root@qdrant-compute:/mnt/qdrant_dev# ls -l
total 8
drwxr-xr-x 3 root root 4096 Mar  2 08:57 demo
drwxr-xr-x 6 root root 4096 Mar  5 06:55 storage
root@qdrant-compute:/mnt/qdrant_dev#
```

Step 9 | Deploy Qdrant Container as a Dev VectorDB on `/mnt/qdrant_dev`

Run Qdrant using the official Docker image without specifying a tag, which pulls the latest version by default. The **/mnt/qdrant_dev** mount is based on the FlexClone created from the production Qdrant volume and is used for Development purposes.

```
root@qdrant-compute:~# docker run -d   --name qdrant-dev   -p 6335:6333   -p 6336:6334   -e
QDRANT__STORAGE__ON_DISK_PAYLOAD=true   -v /mnt/qdrant_dev/storage:/qdrant/storage  qdrant/qdrant
aa119fec70252564bb884c855fb0c32148ac1bf16d541a1b869d15dca8d749fe
```

Container Runtime Parameters

- **-d** - Runs the container in Detached Mode
- **--name** - Assigns the container a readable name
- **-p** - Map a port from the host machine to a port inside the container
- **-e** - Sets an environment variable inside the container
- **-v** - Creates a volume mount that maps the host directory

CLI Validation

```
root@qdrant-compute:~# docker ps

CONTAINER ID   IMAGE           COMMAND             CREATED          STATUS          PORTS    NAMES

aa119fec7025   qdrant/qdrant   "./entrypoint.sh"   10 minutes ago   Up 10 minutes   0.0.0.0:6335-
>6333/tcp, [::]:6335->6333/tcp, 0.0.0.0:6336->6334/tcp, [::]:6336->6334/tcp   qdrant-dev
```

Expected Output:

- Image Name = qdrant
- STATUS = Up
- PORTS= 6335->6333/tcp, 6336->6334/tcp

Create a TEST FlexClone volume for Qdrant database

The same procedure used for creating the **DEV FlexClone volume** is followed to provision the **TEST FlexClone volume** and mount it on the Qdrant host.

While executing the steps, replace the DEV-specific identifiers with TEST-specific values to ensure correct configuration:

- Replace **qdrant_dev** with **qdrant_test** (volume name)
- Replace the junction path **/qdrant_dev** with **/qdrant_test**
- Replace the Vultr compute mount point **/mnt/qdrant_dev** with **/mnt/qdrant_test**
- Use unique port mappings for the TEST Qdrant container to avoid conflicts with existing PROD and DEV instances

Ensure these substitutions are applied consistently across all commands and configurations during the FlexClone creation and mount process, and the TEST mount point can then be used to run a Qdrant instance for testing purposes using the dataset available at the time the FlexClone was created.

Install AMD Enterprise AI Suite with Bloom

This section describes how to install the **AMD Enterprise AI Suite,** which provides the infrastructure required to run the **AMD Blueprint "Chat with Your Documents"** application.

The blueprint implements a **Retrieval-Augmented Generation (RAG)** architecture that combines:

- Enterprise document storage
- Vector search for semantic retrieval
- GPU-accelerated inference for large language models

These components run on the **AMD Enterprise AI Suite**, a Kubernetes-based environment optimized for AMD Instinct GPUs. The platform is installed using the **Cluster Bloom installer,** which automates deployment of the Kubernetes cluster, GPU drivers, and AI platform services.

This automated workflow enables a fully functional GPU-enabled AI platform to be deployed quickly with minimal manual configuration.

Download and Prepare the Bloom Installer

Step 1 | Download the Bloom Installer

Download the Bloom installer from the official **Cluster Bloom release repository.**

```
wget https://github.com/silogen/cluster-bloom/releases/download/v1.2.2/bloom
```

Step 2 | Make the Installer Executable

Grant execution permission so the Bloom installer can run as a deployment tool.

```
chmod +x bloom
```

This allows the Bloom installer to run as an executable deployment tool.

Step 3 | Create the Bloom Configuration File

Define the configuration used by Bloom to deploy the Kubernetes platform.

Create the configuration file

```
nano bloom.yaml
```

The **bloom.yaml** file defines the platform deployment configuration, including:

- Domain settings
- Cluster disk configuration
- GPU node configuration
- Authentication parameters
- TLS certificate options

Example Bloom Configuration

Sample configuration **(bloom.yaml)** used to deploy the AMD Enterprise AI Suite.

```
DOMAIN: <YOUR_DOMAIN>.nip.io
OIDC_URL: https://kc.<YOUR_DOMAIN>/realms/airm
FIRST_NODE: true
GPU_NODE: true
CERT_OPTION: generate
USE_CERT_MANAGER: true
CLUSTER_DISKS: </dev/your_disk_or_partition>
CLUSTERFORGE_RELEASE: <CLUSTERFORGE_RELEASE_TARBALL_URL>
NO_DISKS_FOR_CLUSTER: false
```

Below **“bloom.yaml”** configuration file **is used in this build guide** to deploy the AMD Enterprise AI Suite.

```
aimsdemo@phase3-amd:~/workspace/admin$ cat bloom.yaml
CERT_OPTION: generate
CLUSTER_DISKS: /dev/vdb
CLUSTERFORGE_RELEASE: https://github.com/silogen/cluster-forge/releases/download/v1.5.2/release-enterprise-ai-v1.5.2.tar.gz
CONTROL_PLANE: false
DOMAIN: 192.0.2.40.nip.io
FIRST_NODE: true
GPU_NODE: true
NO_DISKS_FOR_CLUSTER: false
OIDC_URL: https://192.0.2.40.nip.io/realms/airm
USE_CERT_MANAGER: false
```

Configuration Parameters Explanation

Parameter	Description
DOMAIN	Hostname used to access the platform. A nip.io domain automatically resolves the server IP when DNS is not available.
OIDC_URL	OpenID Connect identity provider used for authentication.
FIRST_NODE	Specifies that this node is the first control node of the Kubernetes cluster.
GPU_NODE	Enables GPU acceleration for AI workloads on this node.
CERT_OPTION	Configures TLS certificate generation for secure HTTPS access.
USE_CERT_MANAGER	Enables automated certificate lifecycle management in Kubernetes.
CLUSTER_DISKS	Disk or partition used for cluster storage.
CLUSTERFORGE_RELEASE	URL of the ClusterForge release package used to deploy AMD Enterprise AI components.
NO_DISKS_FOR_CLUSTER	Determines whether disks are automatically configured for cluster storage.

Note: Use the example configuration file “bloom.yml” as a **template**, replace placeholders with values specific to your environment.

Launch the Platform Installation

Run the **Bloom installer** using the configuration file.

```
sudo ./bloom --config bloom.yaml
```

```
aimsdemo@phase3-amd:~/workspace/admin$ sudo ./bloom --config bloom.yaml
[sudo] password for aimsdemo:
🚀 Starting Cluster-Bloom Web Interface...

📄 Configuration file: bloom.yaml
🔧 Pre-filled configuration ready for review and confirmation

🌐 Web interface starting on http://192.0.2.1:62079
🔒 Configuration interface accessible only from localhost
🔧 Configure your cluster at http://192.0.2.1:62079

🔗 For remote access, create an SSH tunnel:
    ssh -L 62079:192.0.2.1:62079 user@remote-server
    Then access: http://192.0.2.1:62079
```

During installation, Bloom automatically performs the following tasks:

- Deploys an **RKE2 Kubernetes cluster**
- Installs **ROCm GPU drivers**
- Configures **GPU runtime support**
- Initializes **cluster storage**
- Deploys **AMD Enterprise AI Suite services**
- Installs platform components required for AI workloads

The installation process typically takes **15–20 minutes,** depending on the system configuration.

Once initialization begins, the installer exposes a **web-based interface** that provides real-time installation monitoring and configuration validation.

Access the Bloom Installer UI

To access the installer interface securely from your local machine, create an **SSH tunnel** to the server.

```
ssh -L 62079:192.0.2.1:62079 <USER>@<SERVER_IP>
```

```
admin:~$ ssh -L 62079:192.0.2.1:62079 aimsdemo@192.0.2.40
Welcome to Ubuntu 24.04.4 LTS (GNU/Linux 6.8.0-101-generic x86_64)
```

Open the installer interface in a browser:

```
http://192.0.2.1:62079
```

The Bloom installer UI provides:

- Installation progress
- Configuration validation
- Deployment confirmation

The interface displays a **pre-populated configuration preview,** allowing users to review and modify settings before initiating the final installation. If no changes are required, click **Generate Configuration** and **Start Installation.**

Installation Best Practice

It is recommended to run the installation in screen or tmux so the deployment survives SSH disconnects.

```
screen -S bloom-install
```

The following figures illustrate the **Bloom installer UI** accessed through an SSH tunnel, where the deployment configuration is automatically populated from the `bloom.yaml` file and can be reviewed or modified prior to installation. The interface provides a structured workflow for validating configuration parameters and initiating the deployment process.

(Due to the length of the interface, representative sections from the beginning and end of the configuration workflow are shown.)

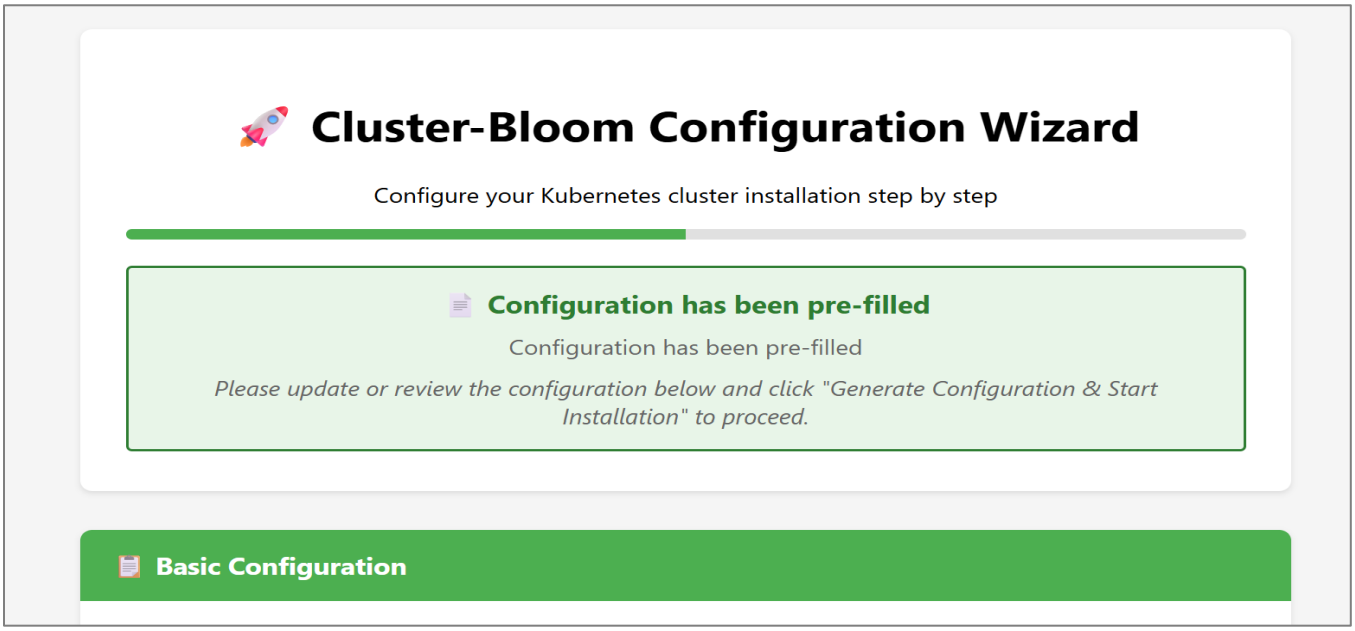

Figure 4 - Bloom installer UI accessed through an SSH tunnel showing pre-populated configuration parameters

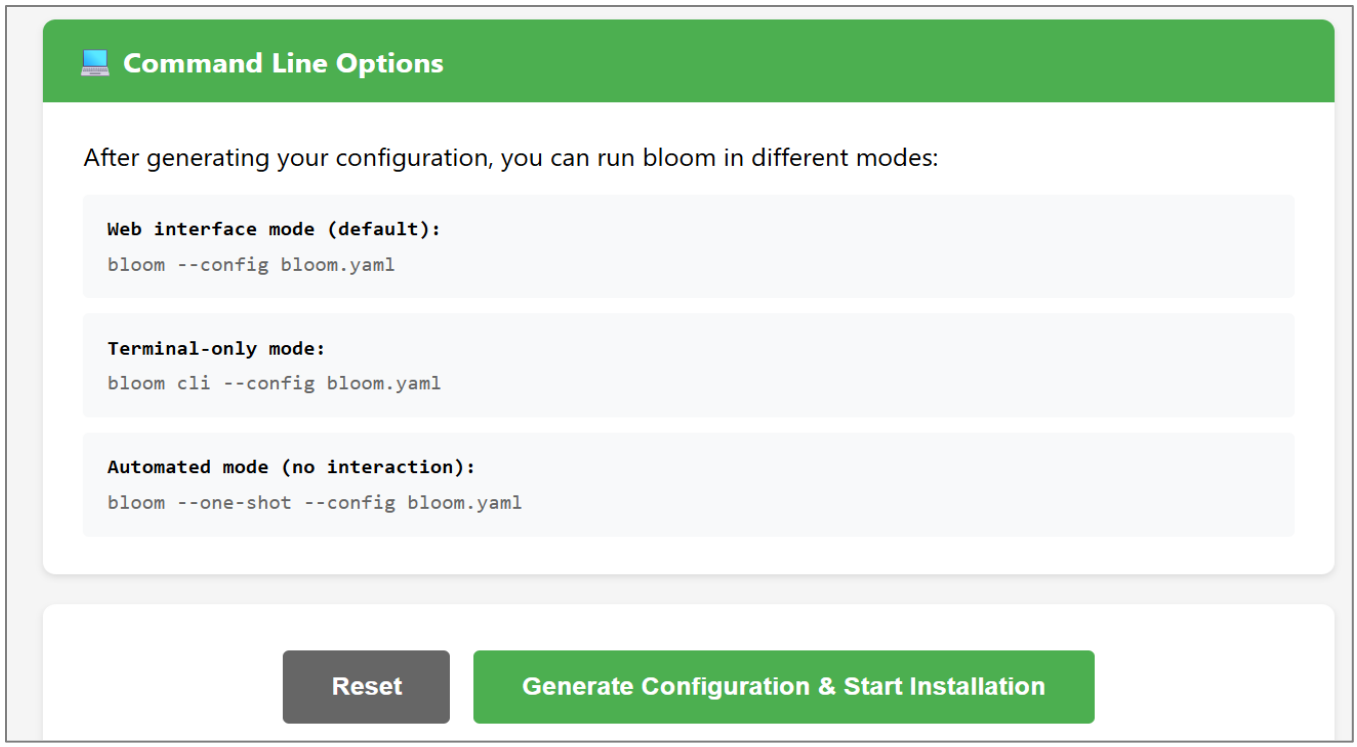

Figure 5 - Bloom installer UI accessed through an SSH tunnel showing deployment execution controls

If no changes are required, click **Generate Configuration and Start Installation** to begin the platform deployment. Once the installation starts, the console displays the initialization logs, which can be used to monitor the progress of the installation.

Following console log shows the installation start.

```
aimsdemo@phase3-amd:~/workspace/admin$ sudo ./bloom --config bloom.yaml
[sudo] password for aimsdemo:
🚀 Starting Cluster-Bloom Web Interface...
📄 Configuration file: bloom.yaml
🔧 Pre-filled configuration ready for review and confirmation
🌐 Web interface starting on http://192.0.2.1:62079
🔒 Configuration interface accessible only from localhost
🔧 Configure your cluster at http://192.0.2.1:62079
🔗 For remote access, create an SSH tunnel:
    ssh -L xxxxx user@remote-server
    Then access: http://192.0.2.1:62079
INFO[0003] Found 1 disks from CLUSTER_DISKS:
  /dev/vdb: 2T     /mnt/disk0
INFO[0004] Found 1 disks from CLUSTER_DISKS:
  /dev/vdb: 2T     /mnt/disk0
☑ Configuration received from web interface
🔄 Starting installation...

INFO[0175] Updated viper with 10 config values from web interface
```

Monitor the installation from the Cluster Bloom UI/interface

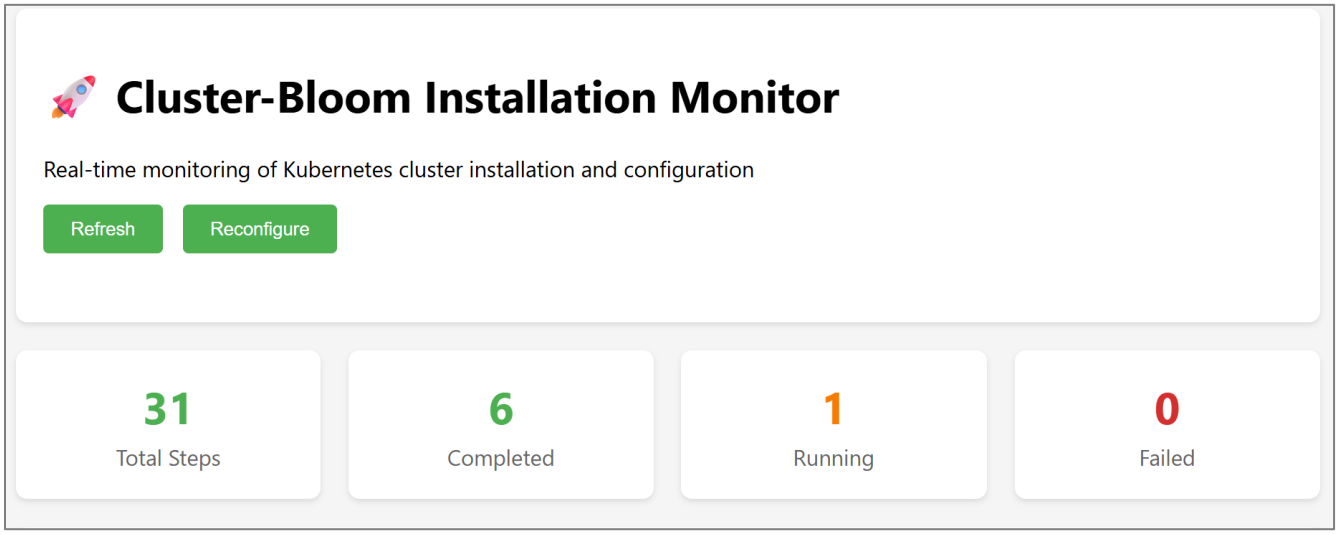

Figure 6 - Cluster Bloom Installation Status (In Progress)

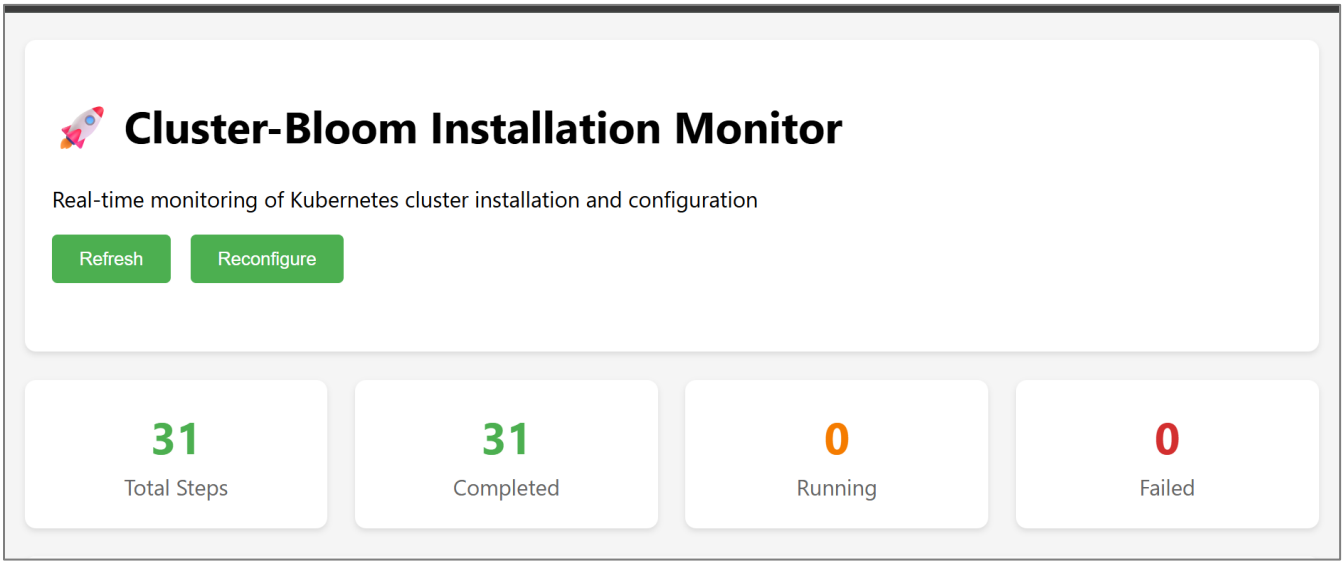

Figure 7 - Cluster Bloom Installation Status (Completed)

The above image shows that all 31 installation steps have completed successfully, with no running or failed tasks, confirming that the installation is complete.

Expected Result

Once the Bloom installation completes successfully:

- The **Kubernetes cluster** is fully operational
- **GPU nodes** are configured with ROCm drivers
- **AMD Enterprise AI Suite services** are deployed

Configure DNS and TLS

To access the **AMD Enterprise AI Platform UI,** DNS and TLS must be configured for the platform ingress. This guide uses a <PUBLIC_IP>.nip.io hostname in the bloom.yaml configuration. The **nip.io** service automatically resolves the hostname to the specified public IP address. In this deployment, the public IP corresponds to the **gateway node that exposes the platform ingress.**

Using nip.io allows the platform services to be accessed through a valid hostname **without requiring external DNS configuration.** This approach simplifies deployments in **lab or test environments** while still providing a stable **HTTPS endpoint.**

If an organization-managed domain is available, replace the nip.io hostname and self-signed certificate with a certificate issued for that domain.

Generate a Wildcard Self-Signed TLS Certificate

Create a wildcard certificate for ***.<PUBLIC_IP>.nip.io.**

First create the OpenSSL configuration file.

```
cat > /tmp/airm-nipio-openssl.cnf <<'CNF'
[req]
default_bits = 2048
prompt = no
default_md = sha256
x509_extensions = v3_req
distinguished_name = dn
[dn]
CN = *.<PUBLIC_IP>.nip.io
[v3_req]
subjectAltName = @alt_names
[alt_names]
DNS.1 = *.<PUBLIC_IP>.nip.io
DNS.2 = <PUBLIC_IP>.nip.io
CNF
```

```
aimsdemo@phase3-amd:~/workspace/generate_tls_cert$ cat > /tmp/airm-nipio-openssl.cnf <<'CNF'
[req]
default_bits = 2048
prompt = no
default_md = sha256
x509_extensions = v3_req
distinguished_name = dn
[dn]
CN = *.192.0.2.40.nip.io
[v3_req]
subjectAltName = @alt_names
[alt_names]
DNS.1 = *.192.0.2.40.nip.io
DNS.2 = 192.0.2.40.nip.io
CNF
```

Generate the TLS certificate and key.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout /tmp/cluster-tls.key \
  -out /tmp/cluster-tls.crt \
  -config /tmp/airm-nipio-openssl.cnf
```

```
aimsdemo@phase3-amd:~/workspace/generate_tls_cert$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout /tmp/cluster-tls.key \
  -out /tmp/cluster-tls.crt \
  -config /tmp/airm-nipio-openssl.cnf
.+..++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*..+.....+........+.+..........++++++
+.+..+.....+.......+..+............+.+.............+....+.........+......++++++++++++++++++++++++++++++++++++++++++++++++
...+...+.+.........+...+.............+.............+...+.+..+....++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++*..+....+.............+......+.........+..+.......+...+...+..+.......+..+..
.......+.........+...+........++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-----
```

Configure TLS

Create the TLS secret used by the **AMD Enterprise AI Suite** ingress to enable HTTPS access for the platform services.

```
kubectl -n kgateway-system create secret tls cluster-tls \
  --cert=/tmp/cluster-tls.crt \
  --key=/tmp/cluster-tls.key
```

```
aimsdemo@phase3-amd:~/workspace/generate_tls_cert$ kubectl -n kgateway-system create secret tls
cluster-tls \
  --cert=/tmp/cluster-tls.crt \
  --key=/tmp/cluster-tls.key

secret/cluster-tls created
```

Restart the HTTPS gateway deployment so the new TLS configuration is applied.

```
kubectl rollout restart deployment/https -n kgateway-system
```

Access the AMD Enterprise AI Suite UI

After the TLS secret is created, access the platform using the hostname configured for your deployment.

- Open the following URL in your browser: *https://airmui.192.0.2.40.nip.io/ (replace with your deployment IP or hostname)*
- Log in using the credentials generated during the platform installation. It is recommended to change the password during the first login.

The following figure shows the AMD Enterprise AI Suite Dashboard.

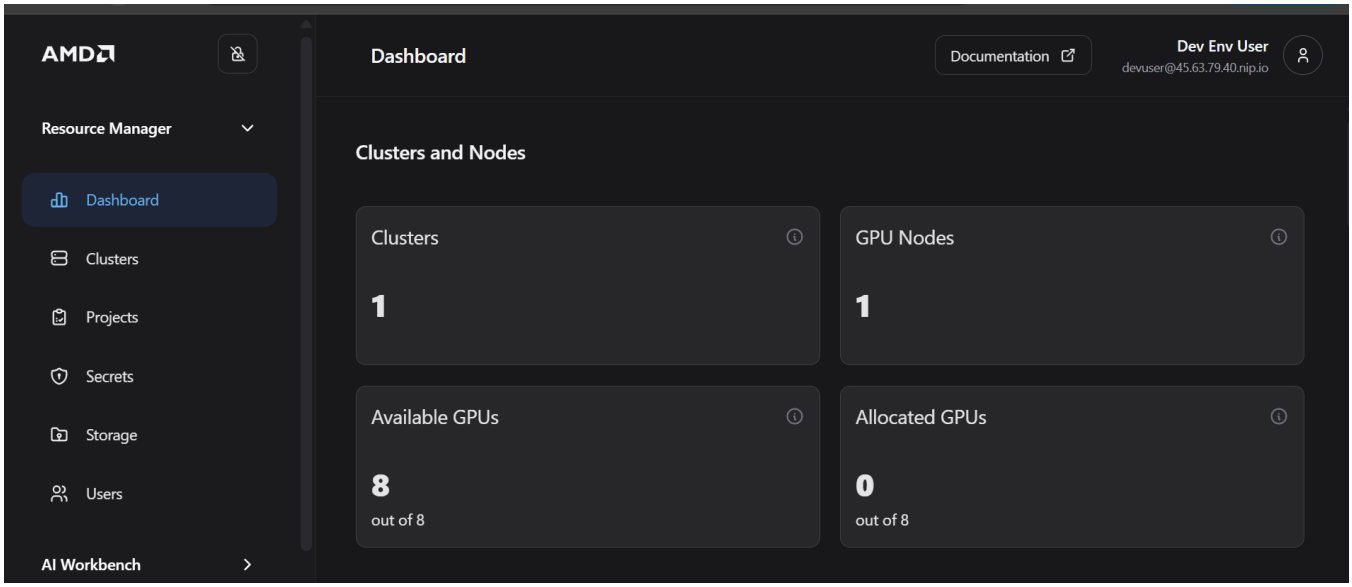

Figure 8 - AMD Enterprise AI Suite Dashboard

Deploy AMD Blueprint with Helm

This section describes how to deploy the **AMD "Chat with Your Documents" blueprint** on the AMD Enterprise AI Suite using Helm.

The blueprint deploys the application services required for a document-based conversational AI system, including the embedding model, large language model (LLM), and vector database integration. By default, the AMD blueprint uses **ChromaDB** as the vector database. In this build guide, the blueprint is customized to use **Qdrant** as the external vector database and **NetApp FlexCache** as the enterprise document source.

The deployment installs the RAG application, embedding service, and LLM inference services on the Kubernetes cluster created during the platform installation.

Modifications to AMD Blueprint: Introducing New Vector DB

The original AMD Blueprint **"Chat with Your Documents"** has been forked, enhanced, and transformed into an enterprise-grade Retrieval-Augmented Generation (RAG) solution named **DocChat**. This upgraded version introduces significant improvements across storage, ingestion, deployment, and scalability.

Key Enhancements

- **Replaced Vector Database:**

  ChromaDB has been replaced with **Qdrant**, providing improved performance, scalability, and production readiness.

- **Enhanced Document Ingestion & Synchronization:**

  The RAG application has been extended to support **automatic document synchronization with FlexCache,** in addition to manual uploads. A **document viewer** has also been integrated to validate and inspect source documents directly within the application.

- **Simplified Deployment Workflow:**

  A unified deployment script, **deploy.sh,** has been introduced to streamline multiple deployment scenarios, including:

  - External or internal Qdrant configurations
  - Local model cache directory usage
  - FlexCache mount integration
  - External exposure via **HttpRoute** with subdomain-based endpoints

- **Deployment Cleanup Utility:**

  A dedicated **clean.sh** script is provided to efficiently remove deployed resources and reset the environment.

- **External Access via HttpRoute:**

  Support for **HttpRoute** has been added to enable secure and flexible external access to the application from the AMD AI Kubernetes cluster.

Clone the Blueprint Repository

Clone the blueprint repository and navigate to the deployment directory.

```
git clone git@github.com:ABC-Company-Pvt-Ltd/solution-blueprint.git
cd solution-blueprints/DocChat
```

Prepare the environment configuration required by the blueprint.

- Rename the environment template file:

  `.env.example → .env`

```
mv.env.example.env
```

- Add the Hugging Face token required by the selected model

```
HF_TOKEN=<your_token>
```

If the model is gated, request access from the model provider before deployment.

During deployment, the Hugging Face token will automatically be stored as a **Kubernetes Secret.**

Review the Model Configuration

The blueprint deploys inference services using **AMD Inference Microservice (AIMs).** The blueprint used in this guide deploys the following models.

Large Language Model (LLM)

- **Model:** meta-llama/llama-3-3-70b-instruct
- **AIM container image:** amdenterpriseai/aim-meta-llama-llama-3-3-70b-instruct:0.8.5-preview
- **Deployment configuration**
    - **GPUs:** 2 GPUs
    - **Precision:** float16

The LLM performs conversational reasoning and response generation for the RAG pipeline.

Embedding Model

- **Embedding model:** intfloat/multilingual-e5-large-instruct
- **Container image:** michaelf34/infinity:0.0.70-amd-gfx942
- **Deployment configuration**
    - GPUs: **1 GPU**

The embedding model generates vector representations for document chunks and user queries.

These parameters can be modified in the **values.yaml** file located in the DocChat directory.

Important parameters include:

- Model ID
- AIMs Docker image
- GPU allocation
- Resource allocation/limits

Review Deployment Parameters

The blueprint repository includes a helper script **deploy.sh** that automates Helm deployment and document ingestion for the Chat with Your Documents application.

Following is the **deploy.sh** which is added as a wrapper to simplify the deployment process.

```
#!/usr/bin/env bash
set -e

# ---- Help ----
if [[ "$1" =~ ^(--help|-h)$ ]]; then
  cat <<EOF
Usage:
  ./deploy.sh --flex-docs-path <path> [options]

Example:
  ./deploy.sh \\
    --flex-docs-path /mnt/<flexcachemountpoint> \\
    --gateway-host my-rag-app.<AIMS_PUBLIC_IP>.nip.io \\
    --qdrant-url http://<QDRANT_HOST>:6333

Options:
  --gateway-host       External hostname (HTTPRoute)
  --qdrant-url         External Qdrant endpoint
  --local-stage-dir    Local staging dir (default: \$HOME/docs)
  --model-cache-path   Model cache path
  --release            Helm release (default: my-rag-app)
  --namespace          Namespace (default: my-namespace)
  --llm-gpus           GPU count

EOF
  exit 0
fi

# ---- Defaults ----
RELEASE="my-rag-app"
NAMESPACE="my-namespace"
VALUES_FILE="values.yaml"
FLEX_TARGET="flex"
FLEX_DOCS_PATH=""
LOCAL_STAGE_DIR="$HOME/docs"
QDRANT_URL=""
MODEL_CACHE_PATH=""
LLM_GPUS=""
GATEWAY_HOST=""

# ---- Parse Args ----
while [[ $# -gt 0 ]]; do
  case "$1" in
    --release) RELEASE="$2"; shift 2 ;;
    --namespace) NAMESPACE="$2"; shift 2 ;;
    --flex-docs-path) FLEX_DOCS_PATH="$2"; shift 2 ;;
    --qdrant-url) QDRANT_URL="$2"; shift 2 ;;
```

```
    --model-cache-path) MODEL_CACHE_PATH="$2"; shift 2 ;;
    --llm-gpus) LLM_GPUS="$2"; shift 2 ;;
    --gateway-host) GATEWAY_HOST="$2"; shift 2 ;;
    --local-stage-dir) LOCAL_STAGE_DIR="$2"; shift 2 ;;
    *) echo "❌ Unknown arg: $1 (use --help)"; exit 1 ;;
  esac
done

# ---- Required Arg ----
[[ -z "$FLEX_DOCS_PATH" ]] && { echo "❌ Missing --flex-docs-path (use --help)"; exit 1; }

APP_NAME="${RELEASE}-aimsb-talk-to-your-documents"

echo "🚀 Deploying: $RELEASE (ns: $NAMESPACE)"

# ---- Namespace ----
kubectl create namespace "$NAMESPACE" --dry-run=client -o yaml | kubectl apply -f -

# ---- Helm Deploy ----
helm dependency update .
helm upgrade --install "$RELEASE" . -n "$NAMESPACE" -f "$VALUES_FILE" \
  ${QDRANT_URL:+--set qdrant.existingService=$QDRANT_URL} \
  ${MODEL_CACHE_PATH:+--set llm.localModelCache.enabled=true \
                       --set llm.localModelCache.hostPath=$MODEL_CACHE_PATH} \
  ${LLM_GPUS:+--set llm.gpus=$LLM_GPUS} \
  ${GATEWAY_HOST:+--set gatewayRoute.host=$GATEWAY_HOST}

# ---- Wait ----
kubectl -n "$NAMESPACE" rollout status deployment/$APP_NAME --timeout=20m

# ---- Access ----
if [[ -n "$GATEWAY_HOST" ]]; then
  BASE_URL="https://$GATEWAY_HOST"
  echo "Using Gateway: $BASE_URL"
else
  echo "🔦 Using port-forward..."
  kubectl -n "$NAMESPACE" port-forward svc/$APP_NAME 17860:80 >/dev/null 2>&1 &
  PF_PID=$!
  sleep 5
  BASE_URL="http://192.0.2.1:17860"
fi

# ---- Sync Docs ----
STAGE_DIR="${LOCAL_STAGE_DIR}/${RELEASE}"
rm -rf "$STAGE_DIR" && mkdir -p "$STAGE_DIR"

echo "📥 Syncing docs..."
rsync -az --include="*/" --include="*.pdf" --include="*.txt" --exclude="*" \
  "${FLEX_TARGET}:${FLEX_DOCS_PATH}/" "$STAGE_DIR/"

DOCS=($(find "$STAGE_DIR" -type f \( -iname "*.pdf" -o -iname "*.txt" \)))
[[ ${#DOCS[@]} -eq 0 ]] && { echo "❌ No docs found"; exit 1; }

echo "📄 Found ${#DOCS[@]} docs"
```

```
# ---- Get Pod ----
APP_POD=$(kubectl -n "$NAMESPACE" get pods -l app=$APP_NAME \
  -o jsonpath='{.items[0].metadata.name}')

# ---- Copy Docs ----
POD_DIR="/tmp/docs"
kubectl -n "$NAMESPACE" exec "$APP_POD" -- mkdir -p "$POD_DIR"

FILES_JSON=""
for f in "${DOCS[@]}"; do
  name=$(basename "$f")
  kubectl -n "$NAMESPACE" cp "$f" "$APP_POD:$POD_DIR/$name"
  FILES_JSON+="\"$POD_DIR/$name\","
done
FILES_JSON="[${FILES_JSON%,}]"

# ---- Index ----
echo "⚡ Indexing..."
curl -s -X POST "$BASE_URL/process" \
  -H "Content-Type: application/json" \
  -d "{\"question\":\"Summarize\",\"files\":$FILES_JSON}" >/dev/null

echo "☑ Deployment + Indexing Complete"

# ---- Cleanup ----
[[ -n "$PF_PID" ]] && kill "$PF_PID" 2>/dev/null || true
```

The script performs the following actions:

- Deploys the Helm chart to the Kubernetes cluster
- Stages documents from the FlexCache mount point
- Indexes documents into the vector database

This automation simplifies the deployment workflow by combining application deployment, document staging, and indexing into a single command.

Vector Database Configuration

The deployment supports both **in-cluster** and **external** Qdrant configurations.

External Qdrant

If a Qdrant endpoint is provided, the internal deployment is skipped, and the application connects to the external qdrant service.

QDRANT_URL environment variable from deploy.sh with default given external QDRANT_URL

```
#!/usr/bin/env bash
set -euo pipefail
SCRIPT_NAME="$(basename "$0")"
RELEASE="my-rag-app"
NAMESPACE="my-namespace"
QDRANT_URL="http://192.0.2.22:6333/"
FLEX_TARGET="flex"
FLEX_DOCS_PATH=""
```

Note: In this build guide, an **external Qdrant instance** is used.

In-cluster Qdrant

If no endpoint is specified, a Qdrant instance is automatically deployed inside the Kubernetes cluster.

If QDRANT_URL environment variable from deploy.sh is empty, Qdrant pod will be deployed from the repo helm chart.

```
#!/usr/bin/env bash
set -euo pipefail
SCRIPT_NAME="$(basename "$0")"
RELEASE="my-rag-app"
NAMESPACE="my-namespace"
QDRANT_URL=""
FLEX_TARGET="flex"
FLEX_DOCS_PATH=""
```

The Qdrant endpoint is configured via the environment variable (**QDRANT_URL**). It is consumed by the following file:

`solution-blueprints/solution-blueprints/DocChat/src/config.py`

```
# Copyright © Advanced Micro Devices, Inc., or its affiliates.
# SPDX-License-Identifier: MIT
import os
import sys
import time
import urllib.parse
import requests
# App Settings
TITLE = "Talk to your documents"
GRADIO_PORT = int(os.getenv("GRADIO_PORT", "7860"))
# Services
INFINITY_EMBEDDING_URL = os.getenv("EMBEDDING_URL", "http://embedding-e5-large:7997/embeddings")
VLLM_BASE_URL = os.getenv("VLLM_URL", "http://llama-3-3-70b:8000/v1")
QDRANT_URL = os.getenv("QDRANT_URL", "")
QDRANT_HOST = os.getenv("QDRANT_HOST", "qdrant")
QDRANT_PORT = int(os.getenv("QDRANT_PORT", "6333"))
```

To enable Qdrant as vector database **values.yaml** is appended with following snippet:

values.yaml

```
qdrant:
  metadata:
    labels: {}
  image:
    repository: "qdrant/qdrant"
    tag: "v1.15.4"
  resources:
    requests: { cpu: "1", memory: "4Gi" }
    limits: { cpu: "2", memory: "8Gi" }
  deployment:
    ports:
      http: 6333
  persistence:
    enabled: true
    size: 25Gi
    storageClassName: mlstorage
  existingService: ""
  fallback:
    enabled: false
  gatewayRoute:
    enabled: true
    host: ""
    gateway:
      name: https
      namespace: kgateway-system
```

View Deployment Script Options

To display all available deployment parameters, run the following command:

```
./deploy.sh --help
```

Output

```
./deploy.sh --help
Usage:
  ./deploy.sh --flex-docs-path <path> [options]

Example:
  ./deploy.sh \
    --flex-docs-path /mnt/<flexcachemountpoint> \
    --gateway-host my-rag-app.<AIMS_PUBLIC_IP>.nip.io \
    --qdrant-url http://<QDRANT_HOST>:6333
Options:
  --gateway-host       External hostname (HTTPRoute)
  --qdrant-url         External Qdrant endpoint
  --local-stage-dir    Local staging dir (default: $HOME/docs)
  --model-cache-path   Model cache path
  --release            Helm release (default: my-rag-app)
  --namespace          Namespace (default: my-namespace)
  --llm-gpus           GPU count
```

The help output below summarizes supported flags, defaults, and examples for the deployment script.

Key Deployment Parameters

The following table summarizes the deployment parameters supported by the **deploy.sh** script.

Options	Description
`--gateway-host`	External hostname (HTTPRoute)
`--qdrant-url`	External Qdrant endpoint
`--local-stage-dir`	Local staging dir (default: `$HOME/docs`)
`--model-cache-path`	Model cache path
`--release`	Helm release (default: `my-rag-app`)
`--namespace`	Namespace (default: `my-namespace`)
`--llm-gpus`	GPU count

Run the Helm Deployment

The **deploy.sh** script supports multiple deployment configurations depending on the environment and infrastructure setup. The following examples illustrate common deployment scenarios.

Deployment Used in This Guide

The following command will deploy the Chatbot application, Embedding model on 1 AMD Instinct GPU, LLM on 2 AMD Instinct GPUs.

Helm Deployment Output

```
./deploy.sh \
  --flex-docs-path /mnt/Flexcache_Site2 \
  --gateway-host my-rag-app.<AIMS_PLATFORM_PUBLIC_IP>.nip.io \
  --qdrant-url http://<Qdrant_IP>:6333/ \
  --local-stage-dir "${HOME}/workspace/Docs" \
  --model-cache-path "${HOME}/workspace/xxxx/model-cache"  # Optional
```

Deployment logs should show the Helm release completed successfully and the application components being created.

Successful Helm release summary

```
aimsdemo@phase3-amd:~/workspace/solution-blueprints/solution-blueprints/DocChat$ ./deploy.sh \
  --flex-docs-path /mnt/Flexcache_Site2 \
  --gateway-host my-rag-app. 192.0.2.40.nip.io \
  --qdrant-url http://192.0.2.22:6333/ \
  --local-stage-dir "${HOME}/workspace/verify/solution-blueprints/solution-
blueprints/DocChat/Docs" \
  --model-cache-path "${HOME}/workspace/xxxx/model-cache"
[INFO] Checking docs path on flex: /mnt/Flexcache_Site2
[INFO] Ensuring namespace exists: my-namespace
[INFO] Creating/updating hf-token secret in namespace: my-namespace
[INFO] Updating helm dependencies
Saving 2 charts
Downloading aimchart-embedding from repo oci://registry-1.docker.io/amdenterpriseai
Pulled: registry-1.docker.io/amdenterpriseai/aimchart-embedding:0.1.2
Digest: sha256:b118d65b79e2d7762b952852042d2c219c7c44f3b4915a98d937e1f2c6aa44b8
Deleting outdated charts
[INFO] Deploying Helm release: my-rag-app
Release "my-rag-app" does not exist. Installing it now.
NAME: my-rag-app
LAST DEPLOYED: Sun Mar  8 09:45:38 2026
NAMESPACE: my-namespace
STATUS: deployed
REVISION: 1
DESCRIPTION: Install complete
TEST SUITE: None
[INFO] Waiting for deployment rollout: my-rag-app-aimsb-talk-to-your-documents
Waiting for deployment "my-rag-app-aimsb-talk-to-your-documents" rollout to finish: 0 of 1 updated
replicas are available...
```

Deployment logs

```
Defaulted container "aimsb-talk-to-your-documents" out of: aimsb-talk-to-your-documents, wait-for-
dependencies (init)
Defaulted container "aimsb-talk-to-your-documents" out of: aimsb-talk-to-your-documents, wait-for-
dependencies (init)
[INFO] Triggering indexing via /process
[INFO] Indexing succeeded.

Deployment completed.
  Release:          my-rag-app
  Namespace:        my-namespace
  Values File:      values.yaml
  App Service:      my-rag-app-aimsb-talk-to-your-documents
  Qdrant:           external (http://192.0.2.22:6333/)
  LLM GPUs:         from values.yaml
  Model Cache Path: /home/aimsdemo/workspace/xxxx/model-cache
  Gateway Host:     my-rag-app.192.0.2.40.nip.io
  Flex Docs Source: flex:/mnt/Flexcache_Site2
  Local Stage Dir:  /home/aimsdemo/workspace/verify/solution-blueprints/solution-
                    blueprints/DocChat/Docs/my-rag-app-my-namespace
  Indexed Files:    17
```

Validate Pod Health

Verify that the application pods are running:

```
kubectl get pods -n my-namespace
```

The application pods show STATUS as Running or Completed.

Application pod status after deployment

```
aimsdemo@phase3-amd:~$ kubectl get pods -n my-namespace
NAME                                                  READY   STATUS    RESTARTS   AGE
llm-my-rag-app-7b5dd8857b-9nwmm                       1/1     Running   0          7h1m
my-rag-app-aimsb-talk-to-your-documents-7f94d48668-wcrdm   1/1     Running   0          7h1m
my-rag-app-embedding-7949f65bb9-76dg8                 1/1     Running   0          7h1m
```

Once the deployment is complete, we can visit following url and get the chat with your document UI:

```
https://my-rag-app.<PUBLIC_IP>.nip.io
```

Different Deployment Scenarios

Basic deployment

Deploys the application and indexes documents from the specified FlexCache path.

```
./deploy.sh --flex-docs-path /mnt/netapp/rag_docs
```

Deployment with external Qdrant

Deploys the blueprint while connecting to an external Qdrant vector database.

```
./deploy.sh \
  --namespace rag \
  --release t2yd \
  --flex-docs-path /netapp/docs \
  --qdrant-url http://<Qdrant_Host_IP>:6333
```

Deployment with local model cache

Mounts a local model cache directory to accelerate model loading.

```
./deploy.sh \
  --flex-docs-path /mnt/Flexcache_Site2 \
  --model-cache-path /mnt/model-cache
```

Validate the Deployment

Verify Qdrant Health and Indexed Content

Open the Qdrant dashboard to verify that the service is reachable and that the target collection exists after document ingestion.

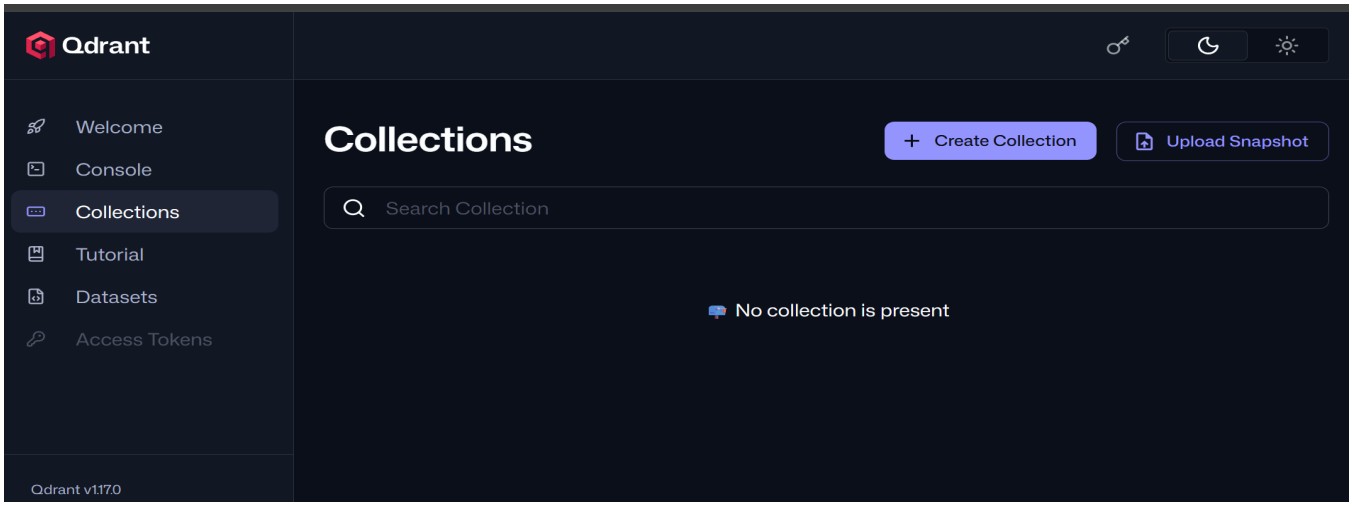

Figure 9 - Initial Qdrant dashboard view

After ingestion completes, verify that the **rag_collection** appears and contains indexed FlexCache documents.

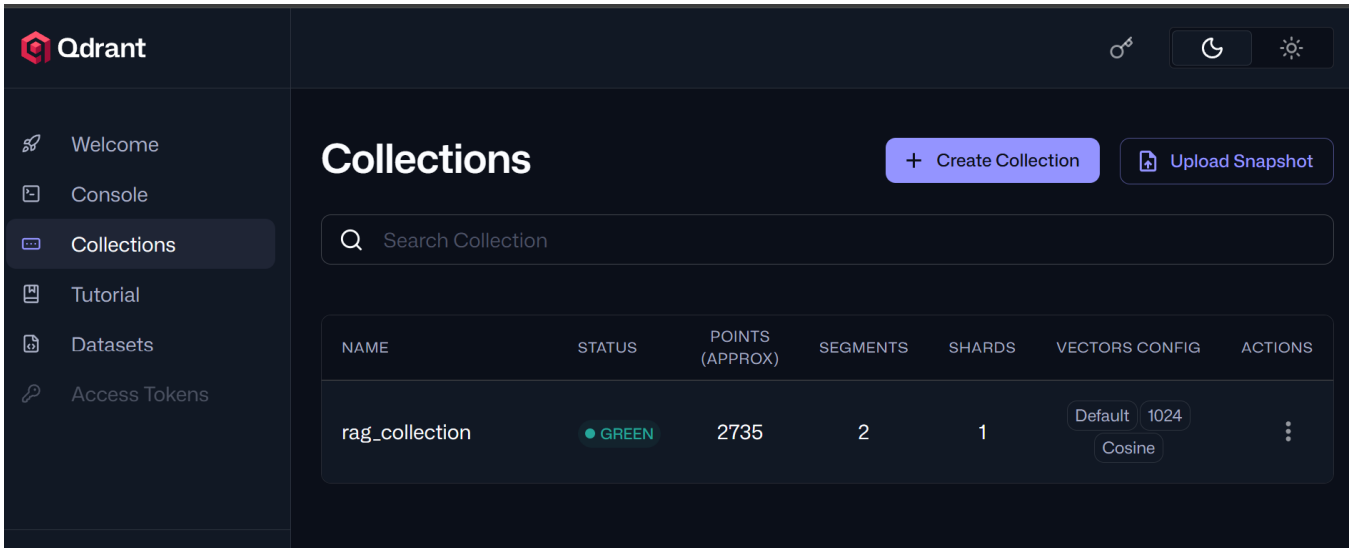

Figure 10 - Qdrant collection populated with indexed FlexCache documents.

Inspect a sample record to confirm that document chunks and metadata were stored correctly in Qdrant.

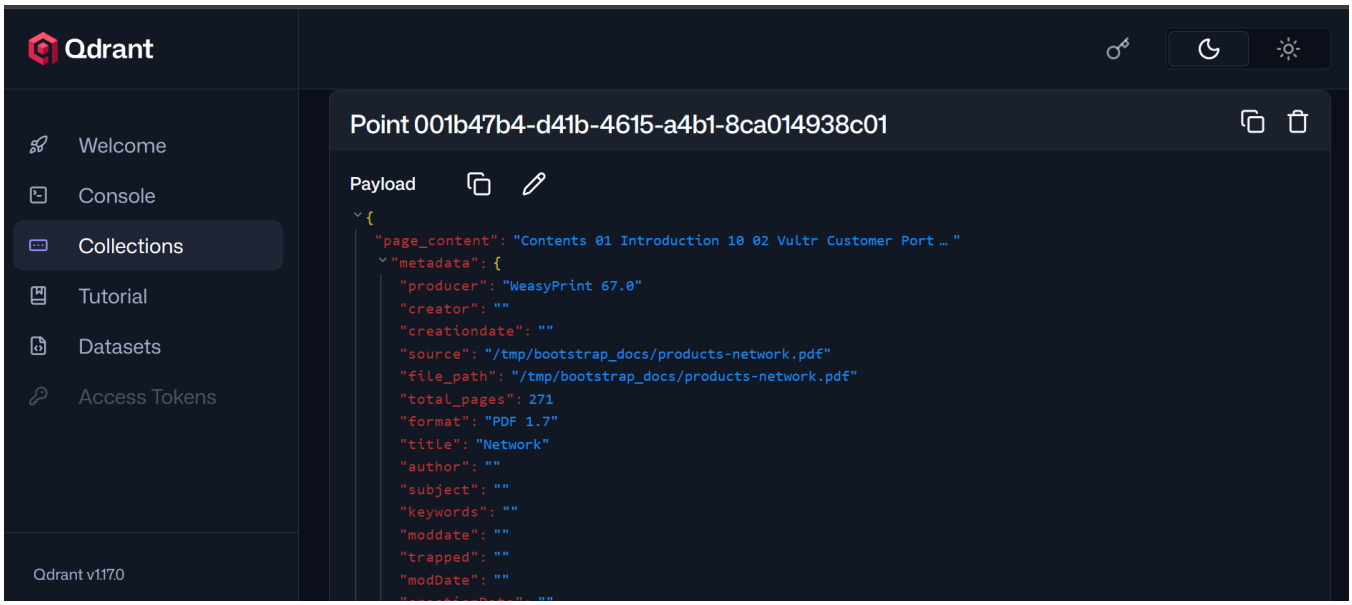

Figure 11 - Sample document chunk stored in Qdrant.

Validate GPU Resource Utilization

This section verifies GPU allocation and memory utilization for the **Chat with Your Documents blueprint** running on **AMD Instinct MI325X GPUs.**

Verify GPU Hardware

Use the following command to display the GPU product name and confirm that the system detects the AMD Instinct GPUs.

```
rocm-smi –showproductname –device 0
```

```
aimsdemo@phase3-amd:~$ rocm-smi --showproductname --device 0

============================ ROCm System Management Interface ============================
======================================== Product Info ========================================
GPU[0]          : Card Series:      AMD Instinct MI325X
GPU[0]          : Card Model:       0x74a5
GPU[0]          : Card Vendor:      Advanced Micro Devices, Inc. [AMD/ATI]
GPU[0]          : Card SKU:         M3250101
GPU[0]          : Subsystem ID:     0x74a5
GPU[0]          : Device Rev:       0x00
GPU[0]          : Node ID:          2
GPU[0]          : GUID:             14054
GPU[0]          : GFX Version:      gfx942
==================================================================================================
====================================== End of ROCm SMI Log =======================================
```

Check Available GPUs

Run the following command to list all GPUs available on the **AMD Instinct MI325X host.**

```
rocm-smi
```

```
aimsdemo@phase3-amd:~$ rocm-smi

============================ ROCm System Management Interface ============================
======================================== Concise Info ========================================
Device  Node  IDs              Temp        Power     Partitions      SCLK    MCLK   Fan  Perf  PwrCap  VRAM%  GPU%
              (DID,   GUID)  (Junction) (Socket)  (Mem,Compute,ID)
0       2     0x74a5, 14054   44.0°C      164.0W    NPS1, SPX, 0    2102Mhz 900Mhz 0%   auto  1000.0W  1%     0%
1       3     0x74a5, 42535   49.0°C      170.0W    NPS1, SPX, 0    2100Mhz 900Mhz 0%   auto  1000.0W  96%    0%
2       4     0x74a5, 21090   45.0°C      156.0W    NPS1, SPX, 0    2103Mhz 900Mhz 0%   auto  1000.0W  96%    0%
3       5     0x74a5, 49827   48.0°C      131.0W    NPS1, SPX, 0    132Mhz  900Mhz 0%   auto  1000.0W  0%     0%
4       6     0x74a5, 13283   40.0°C      127.0W    NPS1, SPX, 0    132Mhz  900Mhz 0%   auto  1000.0W  0%     0%
5       7     0x74a5, 41762   44.0°C      129.0W    NPS1, SPX, 0    132Mhz  900Mhz 0%   auto  1000.0W  0%     0%
6       8     0x74a5, 37216   42.0°C      126.0W    NPS1, SPX, 0    132Mhz  900Mhz 0%   auto  1000.0W  0%     0%
7       9     0x74a5, 417     42.0°C      127.0W    NPS1, SPX, 0    132Mhz  900Mhz 0%   auto  1000.0W  0%     0%

====================================== End of ROCm SMI Log ======================================
```

Check GPU Memory Utilization

The following command displays total and used VRAM on a specific GPU (GPU-0 in this example).

```
rocm-smi –showmeminfo vram –device 0
```

```
aimsdemo@phase3-amd:~$ rocm-smi --showmeminfo vram --device 0
============================ ROCm System Management Interface ============================
================================ Memory Usage (Bytes) ================================
GPU[0]          : VRAM Total Memory (B): 274861129728
GPU[0]          : VRAM Total Used Memory (B): 264046325760
==========================================================================================
====================================== End of ROCm SMI Log ======================================
```

Verify GPU Allocation for the LLM Pod

The following command confirms how many GPUs are assigned to the LLM inference pod.

```
kubectl get pod llm-my-rag-app-7b5dd8857b-9nwmm -n my-namespace -o yaml | grep amd.com/gpu
```

```
aimsdemo@phase3-amd:~$ kubectl get pod llm-my-rag-app-7b5dd8857b-9nwmm -n my-namespace -o yaml |
grep amd.com/gpu
    amd.com/gpu: "2"
    amd.com/gpu: "2"
    amd.com/gpu: "2"
    amd.com/gpu: "2"
    amd.com/gpu: "2"
aimsdemo@phase3-amd:~$
```

Display VRAM Usage in Human-Readable Format

The following command extracts VRAM usage and converts it to gigabytes.

```
rocm-smi --showmeminfo vram --device 0 | awk '/Total Memory|Used Memory/ {printf "%s %.2f
GB\n",$1,$NF/1024/1024/1024}'
```

```
Expected output:

Total VRAM: 255.98 GB

Used VRAM: 245.91 GB
```

Expected GPU Utilization

Memory Utilized on **GPU-0** by Embedding model: intfloat/multilingual-e5-large-instruct

```
aimsdemo@phase3-amd:~$ rocm-smi --showmeminfo vram --device 0 | awk '/Total Memory|Used Memory/
{printf "%s %.2f GB\n",$1,$NF/1024/1024/1024}'
GPU[0] 255.98 GB
GPU[0] 4.45 GB
aimsdemo@phase3-amd:~$
```

Memory Utilized on **GPU-1** by LLM Model: Llama-3.3-70-Instruct

```
aimsdemo@phase3-amd:~$ rocm-smi --showmeminfo vram --device 1 | awk '/Total Memory|Used Memory/
{printf "%s %.2f GB\n",$1,$NF/1024/1024/1024}'
GPU[1] 255.98 GB
GPU[1] 247.13 GB
aimsdemo@phase3-amd:~$
```

Memory Utilized on **GPU-2** by LLM Model: Llama-3.3-70-Instruct

```
aimsdemo@phase3-amd:~$ rocm-smi --showmeminfo vram --device 2 | awk '/Total Memory|Used Memory/
{printf "%s %.2f GB\n",$1,$NF/1024/1024/1024}'
GPU[2] 255.98 GB
GPU[2] 247.13 GB
aimsdemo@phase3-amd:~$
```

Validate the Gradio Application UI

Verify that the **Chat with Your Documents** application is accessible through the web interface.

Open the application in a browser using the following URL:

```
https://my-rag-app.<PUBLIC_IP>.nip.io
```

The application provides a **Gradio-based interface** that allows users to interact with indexed documents through natural-language queries.

Within the interface, users can:

- Select pre-indexed documents from the dropdown list
- Preview documents in the integrated PDF viewer
- Submit queries to retrieve information from indexed content
- Upload PDF or DOC files for ad-hoc conversations

Uploaded documents are automatically processed, embedded, and indexed into **Qdrant collections** for retrieval.

Verify Document Ingestion

Before issuing queries, confirm that document ingestion completed successfully by reviewing the application ingestion logs.

If ingestion fails, verify:

- Qdrant connectivity and service availability
- Embedding service health and GPU allocation
- Correct document path and ingestion configuration

For deployments that ingest documents from a **FlexCache mount point,** ensure password less SSH access is configured between the FlexCache mount node and the Helm deployment node. This allows the deployment script to synchronize documents during setup.

Verify Indexed Documents

Review the indexed documents list to confirm that the synchronized FlexCache content is available to end users.

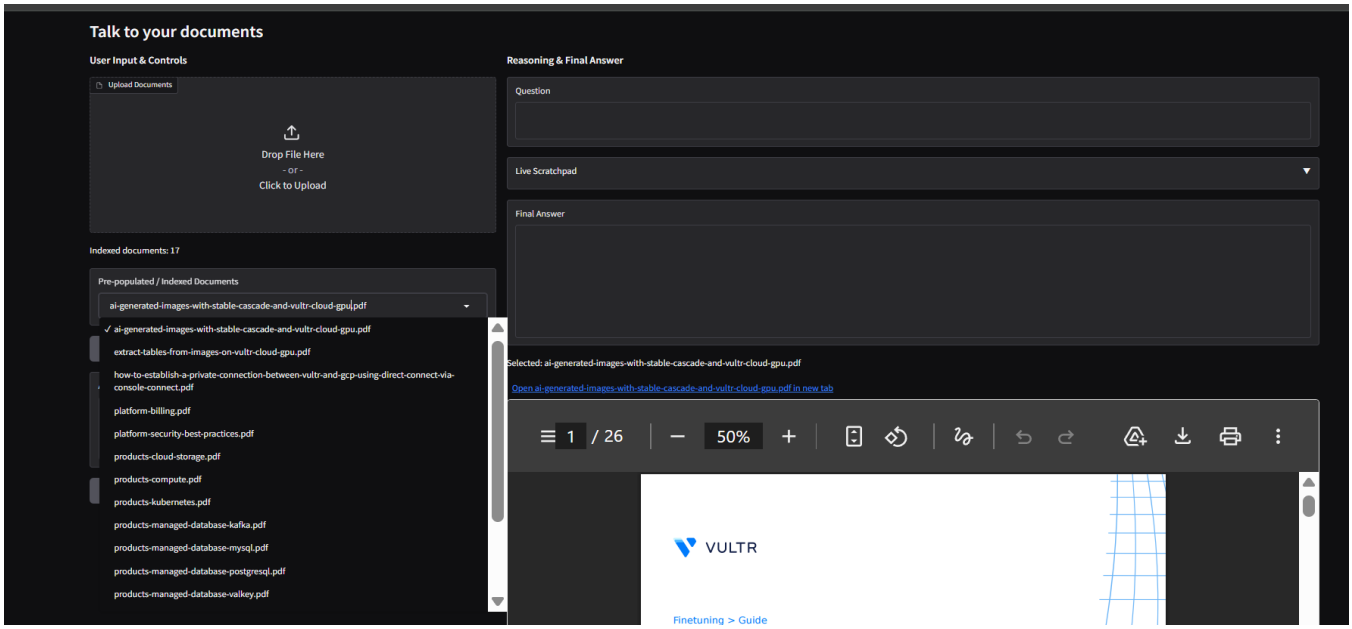

Figure 12 - Indexed documents available in the application

Chat with your Documents and Verify Response

Use the chat panel to query the indexed FlexCache documents and verify that responses are grounded in the retrieved content.

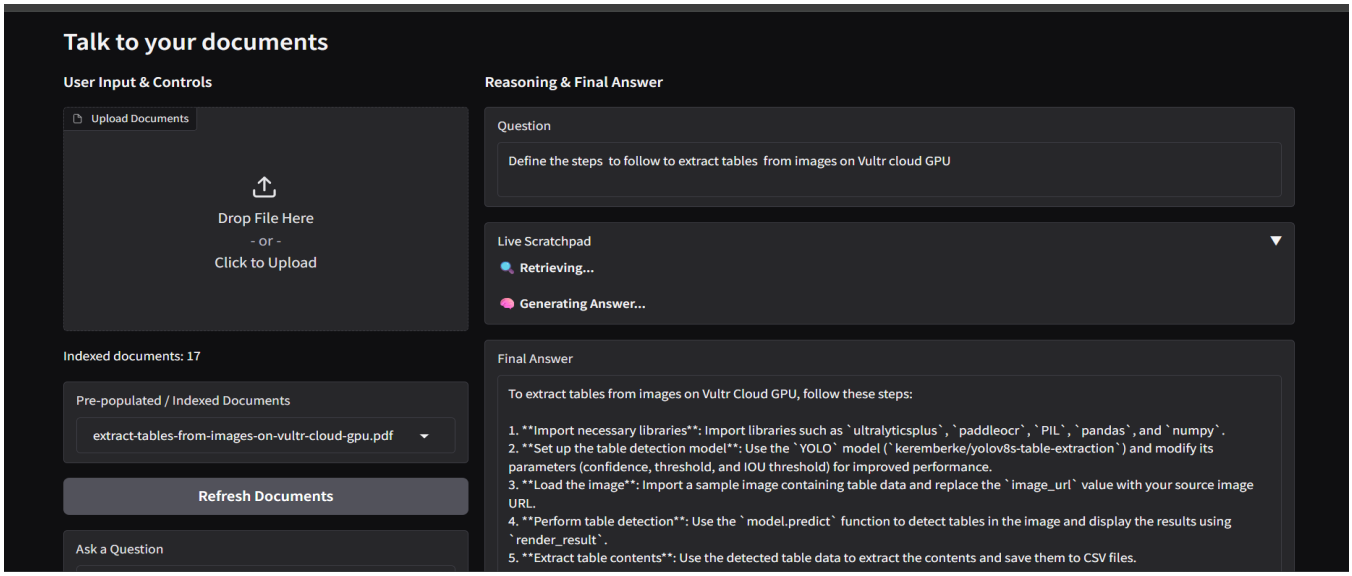

Figure 13 - Chat session using indexed FlexCache documents

Confirm that the document preview matches the source content used for retrieval.

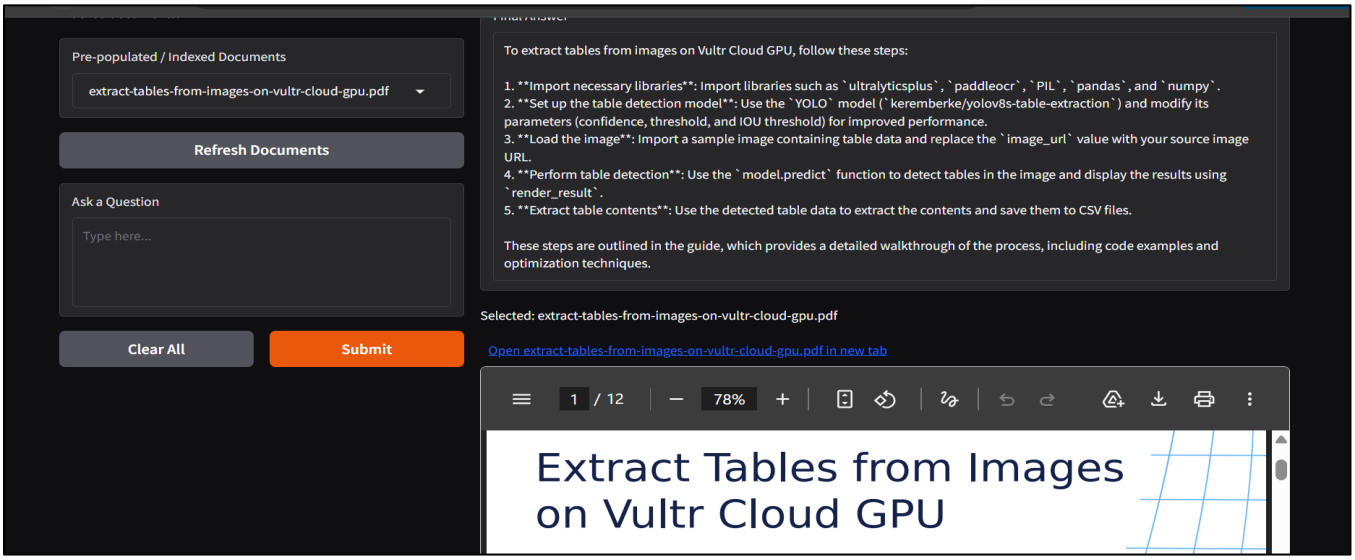

Figure 14 - Document preview for an indexed source file.

Test Ad-hoc Document Upload

Upload an additional file to validate ad-hoc ingestion and conversational querying.

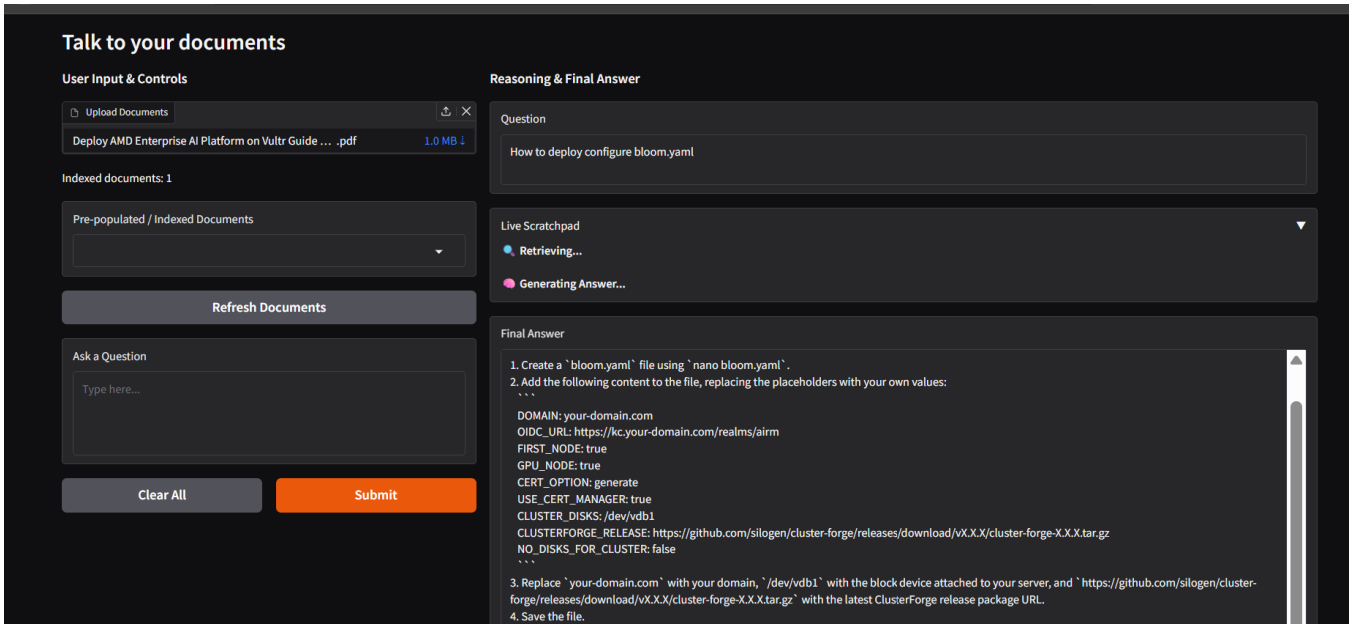

Figure 15 - Upload workflow for a new document.

After uploading, confirm that the viewer renders the new document correctly before sending prompts.

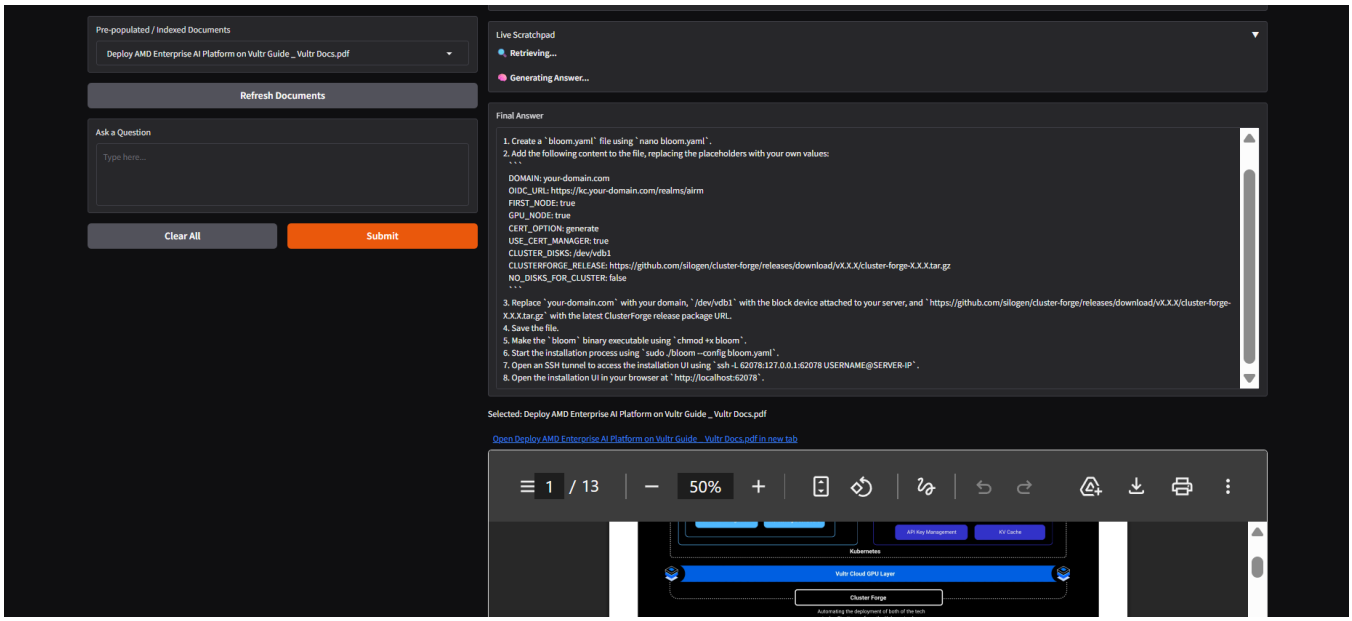

Figure 16 - Preview of the uploaded document.

Troubleshooting and Recovery Checklist

- Qdrant health endpoint returns **HTTP 200: curl http://<QDRANT_IP>:6333/healthz.** If it fails, run docker logs Qdrant and docker restart Qdrant.
- Qdrant storage path exists and is writable. If not, verify mount permissions and ownership for **<QDRANT_MOUNT_DIR>/storage.**
- SSH to FlexCache alias works without password: ssh flex. If it fails, re-run **ssh-copy-id flex** and validate **~/.ssh/config.**
- **rsync** is installed on both nodes and the mount path contains documents. If sync fails, rerun rsync with progress and verify source path permissions.
- Kubernetes pods are running and gateway host resolves the correct public IP. If rollout fails, inspect pod events and use **helm rollback <release> <revision>** to recover the last known good deployment.
- TLS secret exists and is valid: **kubectl -n kgateway-system get secret cluster-tls.**

To Clean the Deployment

```
./clean.sh
```

Cleanup removes the deployed document chat resources when you need to reset the environment.

Following is the clean.sh file used to **clean.sh** the deployment.

```
#!/usr/bin/env bash
set -euo pipefail

SCRIPT_NAME="$(basename "$0")"

RELEASE="my-rag-app"
NAMESPACE="my-namespace"
TIMEOUT="10m"
PURGE_NAMESPACE="false"

usage() {
  cat <<EOF
Usage: $SCRIPT_NAME [options]

Clean deployment resources only (no redeploy).
```

```
Options:
  --release <name>      Helm release name (default: $RELEASE)
  --namespace <ns>      Kubernetes namespace (default: $NAMESPACE)
  --timeout <duration>  Wait timeout for uninstall/delete (default: $TIMEOUT)
  --purge-namespace     Delete namespace after uninstall
  --help                Show this help

Examples:
  $SCRIPT_NAME
  $SCRIPT_NAME --namespace my-namespace --release my-rag-app
  $SCRIPT_NAME --purge-namespace
EOF
}

log() {
  printf '[INFO] %s\n' "$*"
}

fail() {
  printf '[ERROR] %s\n' "$*" >&2
  exit 1
}

require_cmd() {
  command -v "$1" >/dev/null 2>&1 || fail "Missing required command: $1"
}

parse_args() {
  while [[ $# -gt 0 ]]; do
    case "$1" in
      --release)
        [[ $# -ge 2 ]] || fail "--release requires a value"
        RELEASE="$2"
        shift 2
        ;;
      --namespace)
        [[ $# -ge 2 ]] || fail "--namespace requires a value"
        NAMESPACE="$2"
        shift 2
        ;;
      --timeout)
        [[ $# -ge 2 ]] || fail "--timeout requires a value"
        TIMEOUT="$2"
        shift 2
        ;;
      --purge-namespace)
        PURGE_NAMESPACE="true"
        shift
        ;;
      --help|-h)
        usage
        exit 0
        ;;
      *)
        fail "Unknown argument: $1"
```

```
        ;;
    esac
  done
}

uninstall_release() {
  if helm -n "$NAMESPACE" status "$RELEASE" >/dev/null 2>&1; then
    log "Uninstalling Helm release: $RELEASE (namespace: $NAMESPACE)"
    helm -n "$NAMESPACE" uninstall "$RELEASE" --wait --timeout "$TIMEOUT"
  else
    log "Release $RELEASE not found in namespace $NAMESPACE, skipping uninstall"
  fi
}

purge_namespace_if_requested() {
  if [[ "$PURGE_NAMESPACE" != "true" ]]; then
    return 0
  fi
  if kubectl get namespace "$NAMESPACE" >/dev/null 2>&1; then
    log "Deleting namespace: $NAMESPACE"
    kubectl delete namespace "$NAMESPACE" --wait=true --timeout="$TIMEOUT"
  else
    log "Namespace $NAMESPACE does not exist, nothing to delete"
  fi
}

main() {
  parse_args "$@"
  require_cmd helm
  require_cmd kubectl

  uninstall_release
  purge_namespace_if_requested
  log "Cleanup complete"
}

main "$@"
```

```
aimsdemo@phase3-amd:~/workspace/solution-blueprints/solution-blueprints/DocChat$ ./clean.sh
[INFO] Uninstalling Helm release: my-rag-app (namespace: my-namespace)
release "my-rag-app" uninstalled
[INFO] Cleanup complete
```

Conclusion

This guide demonstrated an end-to-end workflow for deploying the **AMD "Chat with Your Documents" blueprint** on the AMD Enterprise AI Suite running on **Vultr Cloud.** The deployment enables organizations to build conversational AI applications that interact with enterprise document repositories while leveraging **FlexCache for hybrid data access,** allowing frequently accessed data to be located closer to AMD GPU resources for AI workloads.

The solution architecture integrates the following components:

- **Hybrid NetApp ONTAP with FlexCache** for governed enterprise document access
- **Qdrant** for semantic indexing and vector similarity search
- **AMD Inference Microservice (AIMs)** for GPU-accelerated embedding and large language model inference
- **AMD Enterprise AI,** which provides the Kubernetes-based platform for orchestrating AI workloads and manages GPU resource
- **Helm,** used to deploy and manage the blueprint application components
- **Vultr Cloud providing AMD Instinct GPU** access for AI workloads

Together, these technologies provide a **scalable architecture for enterprise document intelligence workloads,** enabling AI applications to retrieve relevant information from internal knowledge bases while maintaining existing data governance, storage architecture, and operational controls.