

Accelerate research using NVIDIA AI-Q and NVIDIA RAG Blueprint on IBM Fusion HCI

A reference architecture for on-premises agentic AI research



Table of Contents

- Overview 3
- Executive summary 3
- Scope 4
- Agentic enterprise AI: NVIDIA AI-Q and RAG on IBM Fusion HCI..... 5
- Solution overview 5
- NVIDIA AI-Q and RAG architecture overview 10
 - Core components 10
 - End-to-end system workflows and data flow 12
- Hardware and software stack 13
- Guidelines for deploying on IBM Fusion HCI 14
- Conclusion 18
- Appendix A: Deployment prerequisites 19
- Appendix B: NVIDIA RAG deployment..... 26
- Appendix C: NVIDIA RAG prompt interface 32
- Appendix D: NVIDIA AI-Q deployment..... 36
- Appendix E: NVIDIA AI-Q experience dashboard 40

Overview

This white paper provides a comprehensive technical guide for deploying and integrating the NVIDIA® Retrieval Augmented Generation (RAG) blueprint and the NVIDIA AI-Q Research Agent Blueprint on the IBM® Fusion HCI platform. It presents an overview of the underlying hardware and software stack required for implementation. Deployment guidelines specific to IBM Fusion HCI are described to help technical teams successfully implement these NVIDIA blueprints in their environments. The paper includes appendices that describe deployment prerequisites, step-by-step NVIDIA RAG and AI-Q deployment procedures, and guidance on using the NVIDIA RAG Prompt Interface and NVIDIA AI-Q Research Agent Experience Dashboard.

This technical guide is intended for solution architects, infrastructure engineers, and IT professionals responsible for planning, deploying, and validating AI-powered research assistant workloads on IBM Fusion HCI.

This paper complements, but does not replace, the official NVIDIA and IBM documentation. For additional background, refer to:

- [IBM Fusion Platform documentation](#)
- [NVIDIA RAG Blueprint](#)
- [NVIDIA AI Q Research Agent Blueprint](#)

The solution discussed in this paper is validated on IBM Fusion Platform HCI 2.x with Red Hat® OpenShift®, using NVIDIA NIM™ microservices for large language model (LLM), embedding, and reranking.

Executive summary

Enterprises need to extract actionable intelligence from large volumes of proprietary content—manuals, research papers, service records, and operational data — without exposing sensitive information to public cloud services or accepting the accuracy risks of ungrounded AI models. Traditional search and standalone large language models lack the context, provenance, and governance controls that enterprise deployments require.

This reference architecture addresses that gap by combining three validated components: the NVIDIA RAG Blueprint for grounded, multimodal document retrieval; the NVIDIA AI-Q Research Agent Blueprint for enterprise deep research that provides both quick answers, citations, and in-depth, report-style research in one system, with benchmarks and evaluation harnesses so you can measure quality and improve over time. It also includes IBM Fusion HCI for the GPU-accelerated, OpenShift-based infrastructure needed to run both blueprints together on-premises. Together, these components form a cohesive, enterprise-ready platform for deploying agentic AI research workflows in data-sovereign environments.

This paper also provides deployment guidance for installing and configuring both NVIDIA blueprints on IBM Fusion HCI to support adoption in enterprise environments. The result is a production-ready research assistant that is accurate, explainable, and fully sovereign—compressing hours of manual analysis into minutes while meeting enterprise security and governance requirements.

Scope

This paper defines a focused scope for deploying and validating the NVIDIA RAG Blueprint and NVIDIA AI-Q Research Agent Blueprint on IBM Fusion HCI. The scope clarifies which architectural components, deployment activities, and operational considerations are addressed, and explicitly identifies topics that are outside the intent of this guide.

This paper covers the following areas:

- An overview of the NVIDIA AI-Q Research Agent and NVIDIA RAG architecture
- Deployment guidelines for the solution on IBM Fusion HCI
- Deployment and validation of the NVIDIA RAG Blueprint through Helm on IBM Fusion HCI platform
- Deployment and validation of the NVIDIA AI-Q on top of the existing NVIDIA RAG deployment
- Operational considerations for running these blueprints on IBM Fusion HCI

The following topics are outside the scope of this document:

- Model fine-tuning and prompt engineering
- Custom user interface (UI) development
- Performance benchmarking

Together, these scoped components define a complete, validated reference for deploying an enterprise-grade research assistant on IBM Fusion HCI. The following section introduces the solution architecture in detail—explaining how NVIDIA RAG and NVIDIA AI-Q work together on IBM Fusion HCI to deliver grounded retrieval, agentic reasoning, and production-ready operations.

Agentic enterprise AI: NVIDIA AI-Q and RAG on IBM Fusion HCI

Enterprises today sit on vast quantities of valuable knowledge—spread across documents, reports, images, and data repositories—yet lack the tools to efficiently search, synthesize, and reason across that information at the speed required for making business decisions. The combination of the NVIDIA RAG Blueprint, the NVIDIA AI-Q Blueprint, and IBM Fusion HCI addresses this challenge end-to-end—delivering a fully integrated, on-premises agentic AI solution that can ingest multimodal enterprise content, retrieve relevant knowledge with high precision, and autonomously plan and execute deep research workflows to produce structured, cited, and actionable outputs. This reference architecture focuses on how these capabilities are assembled and operated together as a validated solution stack.

Rather than assembling these capabilities from disparate components, this reference architecture brings them together as a validated, production-ready stack that organizations can deploy with confidence and operate at enterprise scale. By running the full stack on IBM Fusion HCI, the solution maintains data locality while supporting GPU-accelerated inference and cloud-native operational practices. A single, unified platform reduces the infrastructure footprint, simplifies lifecycle management, and eliminates the integration overhead that typically accompanies multi-vendor AI deployments.

The customer benefits of this combined solution are practical and measurable. Research and analysis of workflows that previously required hours of manual effort across multiple systems can be reduced to minutes, with responses grounded in internal data sources and traceable to authoritative documents.

Taken together, these elements define an enterprise-grade agentic AI foundation designed to support grounded retrieval, agentic reasoning, and production-ready operation within existing enterprise infrastructure environments. The resulting capability is designed to be faster to deploy, easier to operate, and built to meet the governance, security, and performance requirements of production AI environments.

Solution overview

This section describes the integrated solution formed by the NVIDIA RAG Blueprint, the NVIDIA AI-Q Blueprint, and IBM Fusion HCI. Together, these components define a validated reference architecture for building and operating an on-premises deep research agent, combining grounded retrieval, agentic reasoning, and GPU-accelerated infrastructure. The following subsections describe each component and its role in the overall solution.

NVIDIA RAG Blueprint

The NVIDIA RAG Blueprint is a production-ready reference architecture for building enterprise-grade retrieval and generation pipelines grounded in an organization's own data. It addresses one of the most persistent challenges in deploying large language models at scale: ensuring that generated responses are accurate, current, and traceable to trusted internal sources. Built with NVIDIA Nemotron models and the NVIDIA NeMo™ Retriever library, the blueprint handles ingestion and extraction of content from multimodal documents—including text, tables, charts, infographics, and audio—and indexes the resulting embeddings into a GPU-accelerated vector database powered by NVIDIA cuVS for low-latency semantic search at scale.

What distinguishes the NVIDIA RAG Blueprint from a basic retrieval implementation is its depth of configurability and enterprise readiness. The pipeline supports advanced retrieval techniques including query decomposition, reranking, and reflection, and is designed to handle millions of documents across formats that would otherwise require separate processing pipelines. Governance, observability, and safety features are built in from the ground up, with telemetry and evaluation frameworks that allow

teams to measure and optimize performance before and after production deployment. The RAG Blueprint also serves as the foundational layer on which more sophisticated agentic systems, including the NVIDIA AI-Q Blueprint, are built.

Key capabilities

The NVIDIA RAG Blueprint provides a set of core capabilities required to build and operate enterprise-grade retrieval-augmented generation pipelines. These capabilities are designed to support accurate, grounded responses at scale while addressing the multimodality, performance, governance, and operational requirements of production enterprise environments.

The following capabilities form the foundation of the NVIDIA RAG Blueprint:

- **Multimodal document intelligence:** Extracts and understands content across text, tables, charts, infographics, and audio—including complex PDF layouts—using NVIDIA Nemotron models with optional optical character recognition (OCR) support, unlocking enterprise content that traditional retrieval systems cannot process.
- **Hybrid vector search:** Combines dense semantic embeddings with sparse keyword search for high-recall, high-precision retrieval across millions of documents, accelerated by NVIDIA cuVS GPU-indexed vector storage.
- **Reranking and reflection:** A reranking step reorders retrieved passages by relevance before generation. An optional reflection capability operates at two stages—first to refine and reformulate the query before retrieval, then again post-generation to verify the grounding of the response against the source context—together reducing hallucinations and improving answer precision.
- **Grounded generation with citations:** Retrieved context and source references are passed directly to LLMs through NVIDIA NIM microservices, producing responses that are traceable, explainable, and auditable—critical for enterprise compliance and trust.
- **Enterprise-grade operations:** Built-in telemetry, observability, evaluation frameworks, content-safety guardrails, and flexible deployment through Docker Compose or Kubernetes (Helm, NIM Operator) provide the operational controls needed for production AI deployments.

NVIDIA AI-Q Blueprint

The NVIDIA AI-Q Blueprint is an open-source reference architecture designed to help organizations build AI agents capable of conducting deep research across enterprise data. Rather than simply retrieving and surfacing information, AI-Q agents are built to reason—planning a research approach, querying multiple data sources in parallel, identifying gaps in the results, and producing structured reports with cited sources. The blueprint supports ingestion of diverse content types including PDFs, images, tables, and unstructured text, and is built around NVIDIA Nemotron models and the NeMo Agent Toolkit for workflow profiling and optimization. Optional integration with the Tavily web search API extends the agent's reach beyond internal data when needed, while human-in-the-loop checkpoints allow users to review and refine the research plan before report generation begins.

Underpinning the NVIDIA AI-Q Blueprint is the NVIDIA RAG Blueprint, which handles the document ingestion and retrieval pipeline that the research agent depends on. The RAG Blueprint processes enterprise content through NVIDIA Nemotron models—extracting text, tables, charts, and graphics from

multimodal documents—and indexes the resulting embeddings in a GPU-accelerated vector database for low-latency semantic search at scale. At query time, AI-Q routes research questions through the RAG service, uses an LLM-as-a-judge step to evaluate the relevance of retrieved results, and triggers web search only as a fallback when internal sources are insufficient. The RAG Blueprint is a hard dependency and must be deployed before the AI-Q layer, which sits on top and provides the planning, reasoning, and synthesis capabilities that elevate the system from a retrieval tool to a functional research agent.

What NVIDIA AI-Q adds to NVIDIA RAG

While the NVIDIA RAG Blueprint provides the foundational capabilities for grounded retrieval and response generation, the NVIDIA AI-Q Blueprint extends this foundation with agentic reasoning and research orchestration. AI-Q builds on RAG to move beyond single-turn question answering, enabling autonomous planning, multi-step reasoning, and structured synthesis of information across large volumes of enterprise data.

Specifically, NVIDIA AI-Q introduces the following capabilities that enhance and extend the NVIDIA RAG Blueprint:

- **Agentic orchestration and task planning:** AI-Q introduces an autonomous research agent that decides how to decompose a query, when to retrieve versus reason, and how to sequence actions—moving beyond passive question answering to goal-directed research execution.
- **Structured research workflows:** Rather than returning raw retrieved content, AI-Q generates coherent, sourced outputs including analyses, comparative summaries, and multi-section reports—reducing hours of manual synthesis to minutes.
- **Intelligent retrieval evaluation:** An LLM-as-a-judge layer assesses the relevance of results returned by the RAG pipeline, triggering web-search fallback only when internal sources are insufficient—improving answer accuracy while maintaining data sovereignty.
- **Advanced reasoning through Nemotron:** Integration with NVIDIA Nemotron model family brings high-accuracy reasoning to complex, multi-hop research tasks that require drawing connections across large volumes of enterprise content.
- **Observability, evaluation, and governance:** Built-in telemetry and profiling through the NeMo Agent Toolkit provide the visibility needed to monitor agent behaviour, measure accuracy, and optimize latency and cost in production deployments.

When deployed on IBM Fusion HCI, NVIDIA AI-Q Blueprint benefits from predictable performance, secure data locality, and seamless scaling—enabling enterprises to confidently operationalize agentic AI research assistants.

IBM Fusion HCI platform

IBM Fusion HCI is a hyperconverged infrastructure appliance that delivers a fully integrated, production-ready OpenShift environment—combining bare-metal compute, software-defined storage, networking, and optional GPU nodes into a single, factory-configured system. Designed to eliminate the deployment complexity that typically accompanies OpenShift on bare metal, IBM Fusion HCI automates cluster provisioning and Day 2 operations, including upgrades, monitoring, and policy management, through operators and GitOps-based tooling. The result is an enterprise platform that can be stood up in hours

instead of weeks, with a unified operational model that spans containerized applications, virtual machines, and AI workloads under a single management plane.

For AI use-cases specifically, IBM Fusion HCI serves as a converged platform for the full AI workload lifecycle—from data ingestion and vector storage through LLM inference and agentic orchestration. It supports the latest NVIDIA GPUs for compute-intensive workloads including model inference, embedding generation, and reranking, and integrates IBM Storage Scale and content-aware storage to provide high-throughput, persistent data services suited to vector indices and large document repositories. Organizations running latency-sensitive or data-sovereign AI workloads benefit from the ability to co-locate data and compute on-premises, avoiding the egress costs and compliance risks associated with cloud-based AI deployments, while retaining the operational consistency of a cloud-native platform.

IBM Fusion HCI as the platform for NVIDIA AI-Q and RAG Blueprints

IBM Fusion HCI provides the infrastructure foundation on which both the NVIDIA RAG Blueprint and the AI-Q Blueprint can be deployed and operated as a cohesive, production-grade system. Its OpenShift-native architecture means that all NVIDIA NIM microservices—including the Nemotron extraction and embedding models, the cuVS-accelerated vector database, and the AI-Q agentic orchestration layer—are deployed as standard containerized workloads, without requiring infrastructure customization. By hosting both the RAG retrieval pipeline and the AI-Q reasoning layer on a single platform, IBM Fusion HCI enforces data locality and security boundaries that are difficult to achieve in distributed or hybrid cloud configurations, making the platform suitable for enterprises with strict data governance requirements.

From an operational standpoint, IBM Fusion HCI provides documented lifecycle management for current supported releases, reducing deployment risk and enabling a repeatable, auditable path to production. IBM Storage Scale integration provides a high-performance, distributed file system required for vector indices and large document repositories. Non-Volatile Memory Express (NVMe) and GPU Direct support help ensure the throughput and latency characteristics that production AI pipelines demand. Built-in backup and restore services—delivered through IBM Fusion HCI operator-driven data services—protect these workloads against data loss, a consideration that is often overlooked in AI blueprint deployments but is critical for production continuity. The platform's GitOps-friendly design further enables standardization across environments, allowing infrastructure and application configuration to be version-controlled and consistently reproduced across development, staging, and production deployments—providing teams the operational confidence to run NVIDIA AI-Q and RAG at enterprise scale.

Why IBM Fusion HCI for NVIDIA AI-Q and RAG

IBM Fusion HCI provides the infrastructure foundation required to deploy and operate the NVIDIA RAG Blueprint and NVIDIA AI-Q Blueprint as a cohesive, production-grade solution. Its OpenShift-native, GPU-accelerated architecture aligns with the operational model of NVIDIA NIM and NeMo microservices, enabling consistent deployment, lifecycle management, and scaling of both retrieval and agentic reasoning components. By hosting data, AI services, and orchestration on a single, integrated platform, IBM Fusion HCI supports data locality, operational consistency, and governance controls that are critical for running agentic AI workloads in enterprise environments.

Key reasons IBM Fusion HCI is well suited for running NVIDIA AI-Q and RAG Blueprints in enterprise environments include the following:

- **Production-ready OpenShift foundation:** Automated cluster deployment and operator-driven Day 2 operations reduce time to production and eliminate the infrastructure complexity that typically delays NVIDIA blueprint deployments.
- **GPU-accelerated compute for the full AI pipeline:** Support for NVIDIA GPUs provides the compute density needed to run NIM microservices for LLM inference, embedding generation, and reranking on a single platform.
- **Converged AI workloads with data locality:** Hosting the RAG retrieval pipeline and AI-Q reasoning layer together on one platform enforces security boundaries and ensures inference always operates against co-located, trusted enterprise data.
- **High-performance, protected storage:** Storage in IBM Fusion HCI can deliver the throughput and low latency required for vector indices and document repositories, supported by operator-driven backup and restore services.
- **Standardized, repeatable operations:** A documented Fusion HCI lifecycle combined with a GitOps-friendly design enables consistent configuration management across development, staging, and production environments.

NVIDIA AI-Q and RAG architecture overview

This section provides a high-level view of the combined architecture that integrates the NVIDIA RAG Pipeline with the NVIDIA AI-Q Research Agent. The architecture illustrates how document ingestion, retrieval, reasoning, and report generation are orchestrated as an end-to-end system, showing the interaction between user interfaces, agentic components, retrieval services, large language models, and supporting infrastructure. The following diagram highlights the key functional layers and data flows that enable grounded question answering and agentic research workflows on a unified platform.

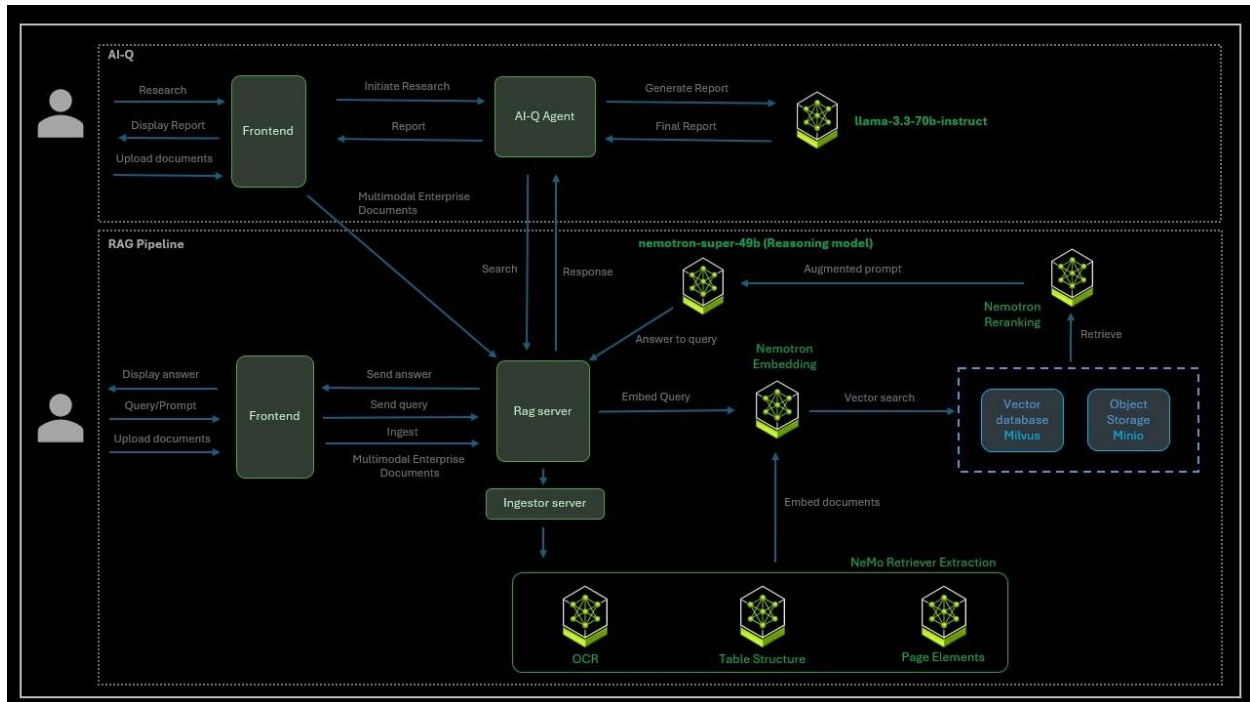


Figure 1: NVIDIA AI-Q and RAG architecture overview

The combined architecture operates across two integrated layers (Figure 1)—the NVIDIA RAG Pipeline, which handles document ingestion and grounded question answering, and the NVIDIA AI-Q, which adds agentic planning and report generation on top of the RAG layer. Together, they form a modular, end-to-end system for transforming enterprise documents into structured, cited research outputs.

Core components

The NVIDIA AI-Q and RAG architecture is composed of several core components that work together to support document ingestion, retrieval, agentic reasoning, and report generation. Each component plays a distinct role in the end-to-end workflow, from user interaction through orchestration, retrieval, and inference, and is designed to operate as part of an integrated, containerized system. The following components represent the primary functional building blocks of the architecture.

Frontend UI: This is the single entry point for users. It supports submitting research queries, uploading enterprise documents, and retrieving generated answers and reports. Routes requests to either the RAG pipeline for direct question answering or the AI-Q agent for deep research workflows.

AI-Q agent: This is the orchestration layer of the research assistant, built on the NVIDIA NeMo Agent Toolkit using LangGraph. It decomposes research topics into subqueries, coordinates parallel searches across the RAG pipeline and optional web sources, applies an LLM-as-a-judge to evaluate relevance of retrieved results, and manages the plan-write-reflect cycle for report generation. It is exposed through a REST API.

NeMo Agent Toolkit: This toolkit provides the LangGraph-based workflow orchestration, observability, telemetry, and LLM configuration management underpinning the AI-Q agent. It translates usage metrics into OpenTelemetry format for integration with standard monitoring tools.

Middleware proxy (nginx): It routes traffic between the frontend, AI-Q backend service, and RAG pipeline services through a single network endpoint, simplifying deployment and inter-service communication.

RAG server: It is the central orchestration service for the retrieval pipeline. It coordinates query embedding, vector search, reranking, and response assembly. It is built on LangChain and exposes OpenAI-compatible APIs for integration flexibility.

Ingestor server: It manages the document ingestion pipeline. It receives uploaded documents from the RAG server and routes them through the Nemotron extraction models before embedding and storing.

Nemotron extraction models: They process multimodal enterprise documents during ingestion. These include nemoretriever-page-elements-v2, which extracts page-level layout elements; nemoretriever-table-structure-v1, which parses tabular data; nemoretriever-graphic-elements-v1, which extracts charts and graphics; and nemotron-ocr which handles text extraction from scanned content.

Embedding (llama-3.2-nv-embedqa-1b-v2): It converts both ingested document content and user queries into dense vector embeddings for semantic search. It is used during ingestion and query execution.

Vector database (cuVS-accelerated): It stores and indexes document embeddings using NVIDIA cuVS GPU-accelerated search. It supports Milvus and Elasticsearch as pluggable back ends, with hybrid dense and sparse search enabled for high-recall retrieval across large document sets.

Reranking (llama-3.2-nv-rerankqa-1b-v2): It reorders retrieved document chunks by relevance before they are passed to the reasoning model, improving answer precision and reducing noise in the generated response.

Reasoning LLM (Llama-3.3-Nemotron-Super-49B-v1.5): It is the primary inference model for the RAG pipeline. It receives an augmented prompt combining the user query and ranked retrieved context, and generates a grounded, factual answer with source traceability.

Report LLM (Llama-3.3-70B-Instruct): It is used exclusively by the AI-Q agent for structured report generation. It produces coherent, multi-section research reports from synthesized research findings, optimized for readability and clarity.

Tavily web search (optional): It supplements internal document retrieval with real-time web search when the RAG pipeline returns insufficient results for a given research query, as determined by the LLM-as-a-judge evaluation step.

End-to-end system workflows and data flow

This section describes the end-to-end workflows and data flows that govern how documents and user queries move through the combined NVIDIA AI-Q and RAG architecture. It outlines how content is ingested, indexed, retrieved, and processed across the system, illustrating the interactions between frontend interfaces, orchestration services, retrieval components, and language models. The following workflows highlight the primary execution paths for document ingestion and query execution.

Document ingestion: The workflow begins when a user uploads documents through the frontend. The RAG server forwards the documents to the Ingestor server, which routes them through the Nemotron models to extract text, tables, graphics, and page elements. The embedding service then converts the extracted content into vector embeddings, which are stored in the vector database.

Query answering (RAG pipeline): The workflow starts when a user submits a query through the front end. The RAG server generates a query embedding, and the vector database performs hybrid semantic search. The reranking service reorders the top results by relevance, after which the ranked context is assembled into an augmented prompt. The reasoning LLM then generates a grounded answer, and the system returns the response with citations to the frontend.

Deep research and report generation (AI-Q): The workflow is initiated when a user submits a research request through the front-end app. The AI-Q agent creates a research plan and decomposes the request into parallel subqueries. Each subquery is sent to the RAG server, where an LLM-as-a-judge evaluates result relevance and triggers Tavily web search as a fallback when required. The system passes synthesized findings to the report LLM, which generates a structured, cited report that is returned to the frontend, with optional human-in-the-loop review and editing at the planning and reporting stages.

Hardware and software stack

This section describes the layered hardware and software stack that supports deployment of the NVIDIA RAG Blueprint and NVIDIA AI-Q Blueprint on IBM Fusion HCI. It outlines how application services, AI microservices, platform software, and underlying infrastructure components are organized across the stack to provide a scalable, GPU-accelerated, OpenShift-based runtime environment. The stack highlights the separation of control plane, CPU worker, and GPU worker roles, along with the integration of storage and platform services required for production AI workloads.

The following diagram displays the **hardware and software stack of the solution**.

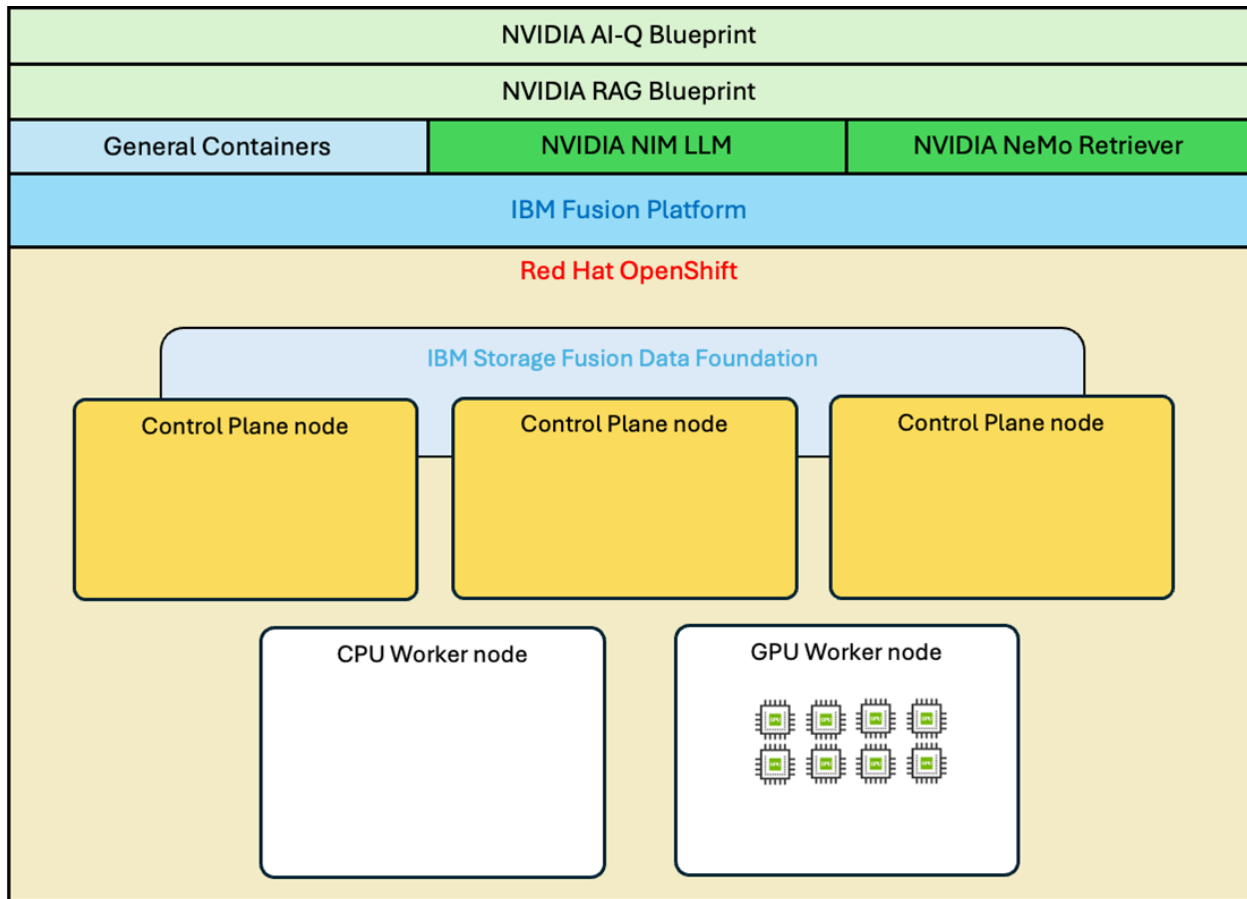


Figure 2: Hardware and Software stack

NVIDIA AI-Q Blueprint: The NVIDIA AI-Q Blueprint provides the reference design for the AI-Q research agent, defining how LLMs, agents, workflows, and retrieval pipelines are assembled. It is deployed as an OpenShift-native solution using operators and containers orchestrated across CPU and GPU worker nodes on IBM Fusion HCI.

NVIDIA RAG Blueprint: The NVIDIA RAG Blueprint defines NVIDIA-recommended architecture for retrieval augmented generation, including embeddings, vector search, and ranking. Its services run as OpenShift workloads, leveraging GPU nodes for embeddings and CPU nodes for retrieval workflows.

General containers: These containers include both general-purpose containers and specialized NVIDIA services.

NVIDIA NIM LLM: It provides GPU-optimized LLM inference microservices, such as Llama and Nemotron. Its pods are scheduled onto GPU worker nodes using the OpenShift GPU Operators and NVIDIA drivers.

NVIDIA NeMo Retriever: It provides embedding generation, retrieval, vector search, and reranking models. Its embedding models run on GPU nodes, while retrieval and search models run on CPU worker nodes and scale elastically across the cluster.

IBM Fusion HCI platform: It provides automation, networking, storage integration, and system lifecycle management. It operates natively as the management foundation for the OpenShift cluster, enabling reliable deployment, upgrades, and observability.

Red Hat OpenShift: It is the Kubernetes foundation for all AI workloads.

IBM Fusion Data Foundation: It provides persistent storage with three-way replication, ensuring data durability and high availability.

Control plane nodes: These nodes are responsible for OpenShift cluster management, including the API server, scheduler, etcd, and controllers. In some configurations, IBM Fusion HCI systems combine control plane and worker nodes.

CPU worker nodes: These nodes provide general compute capacity and run non-GPU intensive services, such as NVIDIA AI-Q orchestration, APIs, data preparation, and retrieval components.

GPU worker node: These nodes are designed for LLM inference, embedding generation, and GPU-accelerated RAG operations.

Guidelines for deploying on IBM Fusion HCI

This section provides practical guidance for deploying the NVIDIA RAG Blueprint and the NVIDIA AI-Q Blueprint on IBM Fusion HCI. It outlines recommended hardware configurations, platform prerequisites, and deployment considerations required to support production-grade agentic AI workloads. The guidance focuses on validated configurations and best practices to help teams plan, size, and deploy the solution consistently and reliably.

Recommended hardware

The recommended hardware configuration is based on the requirements of the combined NVIDIA RAG and AI-Q deployment, including support for GPU-accelerated inference, scalable retrieval pipelines, and OpenShift-based operations. Hardware sizing considerations vary depending on model selection, workload scale, and whether GPU sharing technologies such as NVIDIA MIG or time-slicing are used. The following sections outline validated IBM Fusion HCI configurations and minimum GPU requirements for typical deployment scenarios.

IBM Fusion HCI cluster

The IBM Fusion HCI is a hyperconverged infrastructure solution that integrates compute, storage, and networking resources into a single, scalable platform. The IBM Fusion HCI is built using two OEM hardware configurations.

Supported hardware configurations include:

- Dell hardware—9155-D10 and 9155-D14. For configuration details, see the [Dell hardware documentation](#).
- Lenovo hardware—9155-C10 and 9155-C14. For configuration details, see the [Lenovo hardware documentation](#).

GPU sizing considerations

GPU requirements depend on the selected large language models, workload concurrency, and whether GPU sharing technologies such as NVIDIA MIG or time-slicing are enabled. The following table summarizes the validated minimum GPU requirements for a combined NVIDIA RAG and AI-Q deployment.

Minimum GPU count requirement:

Configuration type	Minimum GPUs required
Without using NVIDIA MIG sharing or time-slicing	10 GPUs (8 for RAG and 2 for AI-Q)
With NVIDIA MIG Sharing or time-slicing	5 to 6 GPUs (varies by LLM Size and GPU type)

Supported GPU types and memory requirements based on model Scale for NIM LLM

GPU selection for the NVIDIA RAG Blueprint and NVIDIA AI-Q Blueprint depends on the scale of the selected large language models and their corresponding memory requirements. The tables below summarize validated GPU types and minimum GPU memory requirements for typical model sizes supported on IBM Fusion HCI.

For RAG Blueprint deployment:

Model type	Model name	Supported GPU type on IBM Fusion HCI	Required GPU memory
Large models (≈49B–70B parameters)	Llama3.3NemotronSuper 49Bv1.5	H200 / RTX Pro 6000	80 GB
Small models (≈3B–8B parameters)	Llama3.1NemotronNano 8Bv1	RTX Pro 6000	45 GB

For AI-Q Research Assistant Blueprint deployment:

Model type	Model name	Supported GPU type on IBM Fusion HCI	Required GPU memory
Large models (≈49B–70B parameters)	Llama3.370BInstruct	H200 / RTX Pro 6000	80 GB
Small models (≈3B–8B parameters)	Llama3.23BInstruct	RTX Pro 6000	25 GB

There are multiple configurations available based on the requirement. Following are a few guidelines (not exhaustive) to select the important hardware components of IBM Fusion HCI.

Entry-level option: In this hardware configuration, three IBM Fusion HCI nodes are configured as both control plane nodes and worker nodes, and one GPU worker node with eight GPUs. With GPU time slicing or NVIDIA MIG sharing, eight GPUs on a single node should be sufficient for RAG and AI-Q Blueprint. With neither GPU time slicing nor NVIDIA MIG sharing, an additional GPU node with two GPUs is required.

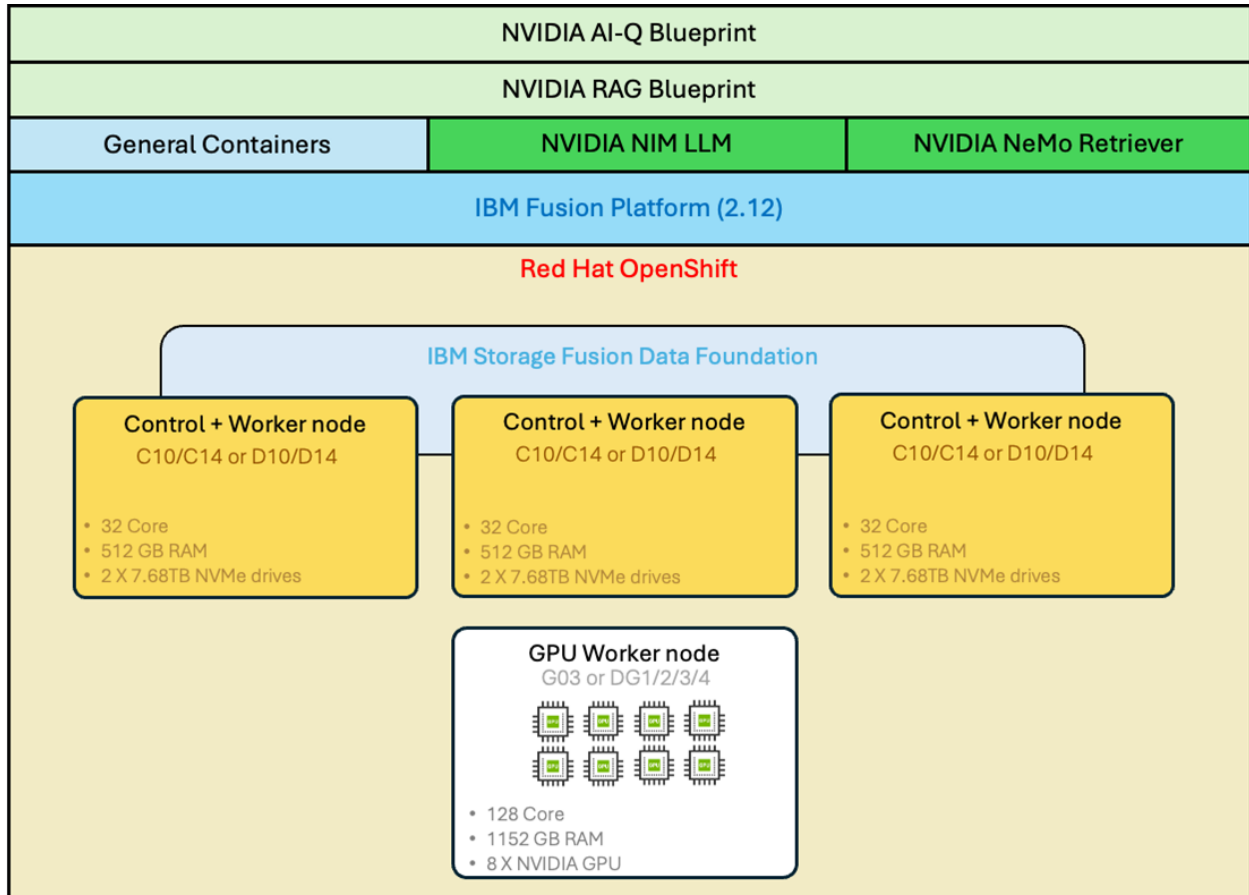


Figure 3: Control plane node + worker node

Scalable option: In this hardware configuration, the Red Hat OpenShift controller and worker nodes are disaggregated to provide better scalability. There are three dedicated control plane nodes and two GPU nodes to support NVIDIA AI-Q and RAG separately. The number of additional worker nodes depends on the application workload requirements.

With no GPU time slicing or NVIDIA MIG sharing, there is a need for additional GPUs in the nodes.

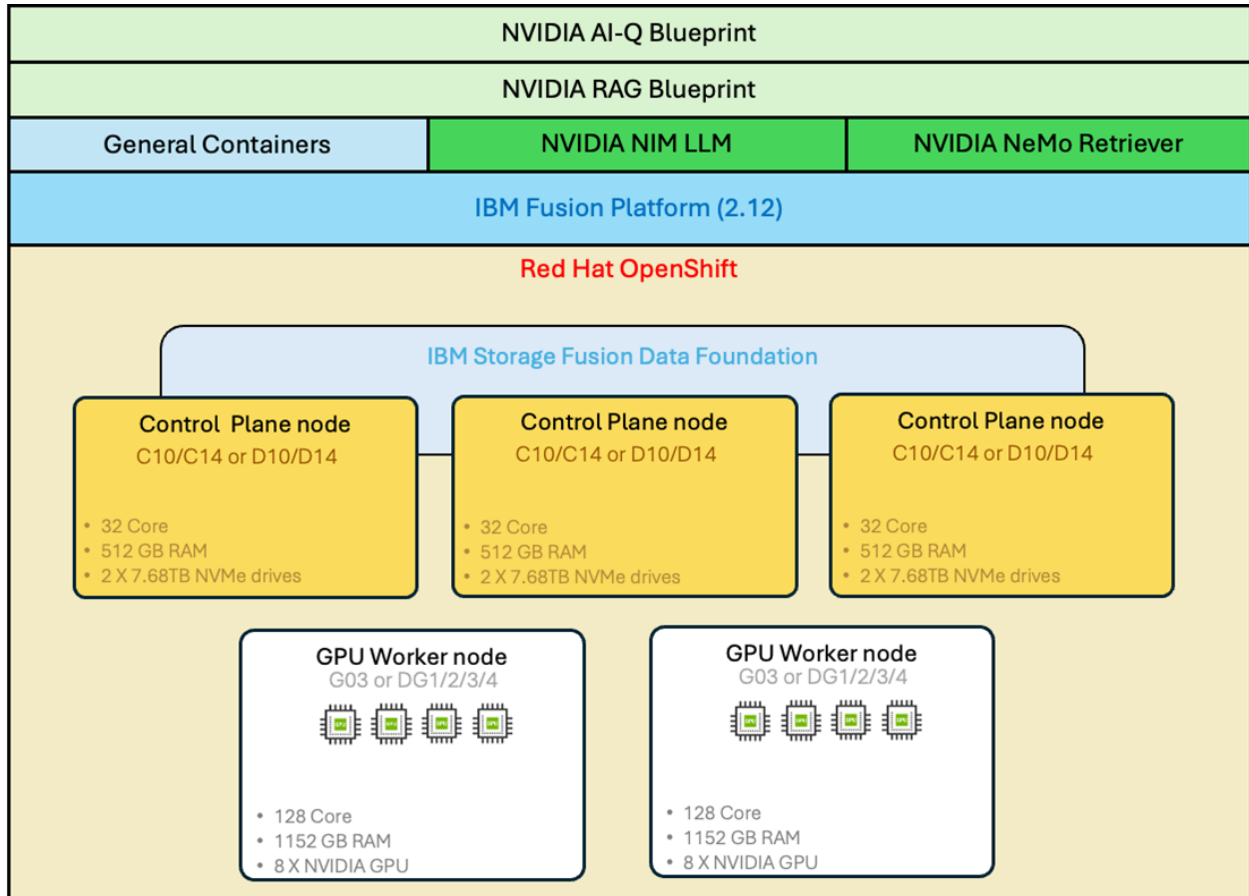


Figure 4: Separate control plane

Platform, software, and access

This subsection outlines the platform prerequisites, software components, and access requirements needed before deploying the NVIDIA RAG Blueprint and NVIDIA AI-Q Blueprint on IBM Fusion HCI. These requirements ensure that the target environment is prepared for installation, configuration, and validation of the solution components.

Before beginning deployment, ensure that the following platform, software, and access prerequisites are met:

- IBM Fusion HCI cluster is installed and operational.
- NVIDIA RAG Helm chart version 2.3.0 is available.
- NVIDIA AI-Q Helm chart version 1.2.0 is available.
- Red Hat OpenShift command-line interface (CLI) tool (oc) and Helm version 3 are installed and accessible.
- Access to the **NVIDIA NGC** registry using an API key (create at <https://ngc.nvidia.com/setup/api-key>).
- Optional: **Tavily API key** for enabling web search functionality (<https://app.tavily.com>).
- OpenShift worker nodes are available with supported NVIDIA GPUs.
- **NVIDIA GPU Operator** is installed and validated on the cluster.

Refer to the appendix section for detailed deployment steps on IBM Fusion HCI.

Conclusion

By combining the NVIDIA RAG Blueprint, the NVIDIA AI-Q Blueprint, and IBM Fusion HCI, organizations can deploy a production-ready agentic research assistant that delivers grounded, explainable answers from trusted internal sources and synthesizes complex, multi-document research into structured, cited outputs.

As demonstrated throughout this paper, IBM Fusion HCI GPU-accelerated OpenShift foundation, high-performance storage, and operator-driven lifecycle management provide the operational backbone needed to scale confidently from pilot to production—while keeping sensitive enterprise data fully on-premises and under enterprise governance.

Appendix A: Deployment prerequisites

This section outlines the prerequisite conditions that must be met before deploying the NVIDIA RAG Blueprint and NVIDIA AI-Q Blueprint on IBM Fusion HCI. It covers required platform configurations, storage setup, GPU operator validation, and node-level checks needed to ensure the environment is prepared for a successful installation. These prerequisites should be completed or verified prior to executing the deployment procedures described in subsequent appendices.

Storage configuration

The storage class is already created or identified for use. Either set the storage class as the default storage class or update the Helm chart with the storage class.

Validate GPU Operator

Verify that the GPU operator is present

```
$ oc get operator | grep nvidia  
gpu-operator-certified.nvidia-gpu-operator
```

Check the availability of the GPUs. First identify the GPU nodes by running the following command.

```
$ oc -n nvidia-gpu-operator get pod -o wide | grep nvidia-driver-daemonset
```

Run the following command for each GPU node.

```
$ oc -n nvidia-gpu-operator exec -it nvidia-driver-daemonset-418.94.202511041748-0-btw8g --  
nvidia-smi
```

The following is the sample output of the nvidia-smi command.

```
% oc -n nvidia-gpu-operator exec -it nvidia-driver-daemonset-418.94.202506251005-0-4xsw9 -- nvidia-smi
```

```
Mon Feb 9 10:34:55 2026
```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+									
NVIDIA-SMI		Driver Version: 580.95.05				CUDA Version: 13.0			
GPU	Name	Perf	Persistence-M	Bus-Id	Disp.A	Memory-Usage	GPU-Util	Uncorr. Compute M.	ECC
Fan	Temp		Pwr:Usage/Cap					MIG	
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+									
0	NVIDIA RTX PRO 6000	Blac...	On	00000000:03:00.0	Off	5523MiB / 97887MiB	0%	Default	0
N/A	33C	P0	90W / 600W					Disabled	
1	NVIDIA RTX PRO 6000	Blac...	On	00000000:04:00.0	Off	91881MiB / 97887MiB	0%	Default	0
N/A	33C	P0	91W / 600W					Disabled	
2	NVIDIA RTX PRO 6000	Blac...	On	00000000:73:00.0	Off	15627MiB / 97887MiB	0%	Default	0
N/A	34C	P0	92W / 600W					Disabled	
3	NVIDIA RTX PRO 6000	Blac...	On	00000000:74:00.0	Off	5833MiB / 97887MiB	0%	Default	0
N/A	32C	P0	87W / 600W					Disabled	
4	NVIDIA RTX PRO 6000	Blac...	On	00000000:83:00.0	Off	3699MiB / 97887MiB	0%	Default	0
N/A	33C	P0	94W / 600W					Disabled	
5	NVIDIA RTX PRO 6000	Blac...	On	00000000:84:00.0	Off	6523MiB / 97887MiB	0%	Default	0
N/A	32C	P0	88W / 600W					Disabled	
6	NVIDIA RTX PRO 6000	Blac...	On	00000000:F3:00.0	Off	11690MiB / 97887MiB	0%	Default	0
N/A	33C	P0	89W / 600W					Disabled	
7	NVIDIA RTX PRO 6000	Blac...	On	00000000:F4:00.0	Off	91799MiB / 97887MiB	0%	Default	0
N/A	33C	P0	95W / 600W					Disabled	
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+									

NVIDIA MIG Sharing

The following are the steps to configure MIG slices for RAG if NVIDIA MIG sharing is enabled.

1. Create a ConfigMap to customize the MIG configuration using the files from the link corresponding to the GPU type.

```
#----- mig-config.yaml -----  
-----  
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: custom-mig-config  
data:  
  config.yaml: |  
    version: v1  
    mig-configs:  
      all-disabled:  
        - devices: all  
          mig-enabled: false  
  
      custom-7x1g10-2x1g20-1x3g40-1x7g80:  
        - devices: [0]  
          mig-enabled: true  
          mig-devices:  
            "1g.10gb": 7  
        - devices: [1]  
          mig-enabled: true  
          mig-devices:  
            "1g.20gb": 2  
            "3g.40gb": 1  
        - devices: [2]  
          mig-enabled: true  
          mig-devices:  
            "7g.80gb": 1
```

2. Apply the spec file to create a ConfigMap.

```
$ oc apply -f mig-config.yaml -n nvidia-gpu-operator
```

3. Label the node with the key value containing the intended MIG configuration.

```
$ oc label node <node name> nvidia.com/mig.config=<mig-config name> --overwrite
```

4. Enable the MIG configuration by applying the following command.

```
$ oc label node <node name> nvidia.com/mig.config=<mig-config name> --overwrite
```

5. Edit the file [values.yaml](#) to configure MIG-optimized resources of RAG Blueprint. If planning to add other values to override the original values in the Helm chart, update the same values.yaml file. The example that follows is for GPU types H100, RTX 6000 Blackwell Pro, and similar GPUs that have 80 GB of memory.

```
#----- overrides.yaml -----  
-----  
  
# MIG-optimized resource configuration for RAG Blueprint  
# This file only overrides GPU resource requirements to use MIG slices  
  
# Ingestor Server - reduce concurrency for MIG  
ingestor-server:  
  envVars:  
    NV_INGEST_FILES_PER_BATCH: "4"  
    NV_INGEST_CONCURRENT_BATCHES: "1"  
  
# NV-Ingest configuration  
nv-ingest:  
  envVars:  
    NV_INGEST_MAX_UTIL: 8  
    MAX_INGEST_PROCESS_WORKERS: 2  
  
# Milvus - uses 1g.10gb MIG slice  
milvus:  
  standalone:  
    resources:  
      limits:  
        nvidia.com/gpu: "0"  
        nvidia.com/mig-1g.10gb: 1  
      requests:  
        nvidia.com/gpu: "0"  
        nvidia.com/mig-1g.10gb: 1  
  
# NV-Ingest NIM Operator overrides  
nimOperator:  
  # Page Elements - uses 1g.10gb  
  page_elements:  
    resources:  
      limits:  
        nvidia.com/gpu: "0"  
        nvidia.com/mig-1g.10gb: 1  
      requests:  
        nvidia.com/gpu: "0"  
        nvidia.com/mig-1g.10gb: 1  
    storage:  
      pvc:  
        storageClass: ""  
  
# Graphic Elements - uses 1g.10gb  
graphic_elements:  
  resources:  
    limits:  
      nvidia.com/gpu: "0"  
      nvidia.com/mig-1g.10gb: 1  
    requests:  
      nvidia.com/gpu: "0"  
      nvidia.com/mig-1g.10gb: 1  
  storage:  
    pvc:
```

```

        storageClass: ""

# Table Structure - uses 1g.10gb
table_structure:
  resources:
    limits:
      nvidia.com/gpu: "0"
      nvidia.com/mig-1g.10gb: 1
    requests:
      nvidia.com/gpu: "0"
      nvidia.com/mig-1g.10gb: 1
    storage:
      pvc:
        storageClass: ""

# OCR - uses 3g.40gb (larger slice)
nemoretriever_ocr_v1:
  resources:
    limits:
      nvidia.com/gpu: "0"
      nvidia.com/mig-3g.40gb: 1
    requests:
      nvidia.com/gpu: "0"
      nvidia.com/mig-3g.40gb: 1
    storage:
      pvc:
        storageClass: ""
# Main NIM Operator overrides for MIG
nimOperator:
# LLM - uses 7g.80gb
nim-llm:
  resources:
    limits:
      nvidia.com/gpu: "0"
      nvidia.com/mig-7g.80gb: 1
    requests:
      nvidia.com/gpu: "0"
      nvidia.com/mig-7g.80gb: 1
    storage:
      pvc:
        storageClass: ""
# Embedding - uses 1g.10gb
nvidia-nim-llama-32-nv-embedqa-1b-v2:
  resources:
    limits:
      nvidia.com/gpu: "0"
      nvidia.com/mig-1g.10gb: 1
    requests:
      nvidia.com/gpu: "0"
      nvidia.com/mig-1g.10gb: 1
    storage:
      pvc:
        storageClass: ""
# Reranking - uses 1g.20gb
nvidia-nim-llama-32-nv-rerankqa-1b-v2:
  resources:
    limits:
      nvidia.com/gpu: "0"
      nvidia.com/mig-1g.20gb: 1
    requests:
      nvidia.com/gpu: "0"
      nvidia.com/mig-1g.20gb: 1

```

```
storage:
  pvc:
    storageClass: ""
```

For more information on MIG configuration, refer to [Creating and applying a custom MIG configuration](#) on NVIDIA Docs Hub.

Enable GPU time slicing

Use the GPU time-slicing option if the GPU does not support MIG sharing and there are limited GPUs or if there is a need to use GPUs efficiently. This configuration advertises multiple logical GPUs to the scheduler. Adjust replicas to match your consolidation goals.

1. Create the ConfigMap.

```
$ cat time-slicing-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: time-slicing-config
data:
  any: |-
    version: v1
    flags:
      migStrategy: none
    sharing:
      timeSlicing:
        renameByDefault: false
        failRequestsGreaterThanOne: false
    resources:
      - name: nvidia.com/gpu
        replicas: 16
```

2. Apply the spec file to create a ConfigMap.

```
$ oc apply -f time-slicing-config.yaml
```

3. Patch the cluster policy to enable GPU time slicing.

```
$ oc patch clusterpolicies.nvidia.com/cluster-policy \ -n nvidia-gpu-operator --type merge \
-p '{"spec": {"devicePlugin": {"config": {"name": "time-slicing-config", "default": "any"}}}}'
```

4. Identify the Daemonset pod running the plugin.

```
$ oc get pod -n nvidia-gpu-operator | grep plugin
nvidia-device-plugin-daemonset-jg8tt          2/2    Running    0          12d
```

5. To restart the device plugin, delete the pod. Verify that the new pod is started and is in the running state.

```
$ oc delete pod nvidia-device-plugin-daemonset-jg8tt -n nvidia-gpu-operator
```

6. Verify allocatable GPUs.

```
$ oc get nodes -o json | jq '.items[].status.allocatable | select(."nvidia.com/gpu" != null)' | jq '. "nvidia.com/gpu"'
```

For more information on GPU time slicing, refer to [Understanding Time-Slicing GPUs](#) on NVIDIA Docs Hub.

Increase PID limits for RAG

1. The RAG deployment needs higher PID limits. Increase the PID limit using the resource KubeletConfig. Create and apply the configuration as shown in the following example.

Warning: Worker nodes reboot after applying this configuration.

```
$ cat <<EOF | oc apply -f -
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-config
spec:
  kubeletConfig:
    podPidsLimit: 12228
  machineConfigPoolSelector:
    matchExpressions:
      - key: machineconfiguration.openshift.io/mco-built-in
        operator: Exists
EOF
```

2. Monitor the machine-config rollout.

```
$ oc get mcp
```

Appendix B: NVIDIA RAG deployment

This section describes the steps required to deploy the NVIDIA RAG Blueprint on IBM Fusion HCI using Helm on Red Hat OpenShift. This appendix covers chart preparation, configuration customization, and installation of the RAG components that provide document ingestion, embedding, retrieval, and grounded generation capabilities. It assumes that all deployment prerequisites outlined in Appendix A have been satisfied and that the RAG deployment will serve as the foundational layer for the NVIDIA AI-Q deployed in subsequent appendices.

The following steps outline the process for downloading, configuring, and deploying the NVIDIA RAG Blueprint on IBM Fusion HCI:

1. Download the NVIDIA RAG Helm chart.

```
$ wget https://helm.ngc.nvidia.com/nvidia/blueprint/charts/nvidia-blueprint-rag-v2.3.0.tgz
```

2. Extract the chart.

```
$ tar xvzf nvidia-blueprint-rag-v2.3.0.tgz
```

3. Create a namespace where all the RAG components will be running.

```
$ oc create namespace rag
```

4. Edit the file `nvidia-blueprint-rag/templates/deployment.yaml` and add the mounts in the `volumeMounts` section as shown in the following example.

```
volumeMounts:
  - name: prompt-volume
    mountPath: /prompt.yaml
    subPath: prompt.yaml
  - name: tmp-data
    mountPath: /workspace/tmp-data
  - name: prom-data
    mountPath: /tmp-data/prom_data
```

5. Also add the corresponding entries in the `volumes` section as shown in the following example.

```
volumes:
  - name: prompt-volume
    configMap:
      name: {{ include "nvidia-blueprint-rag.fullname" . }}-prompt
      defaultMode: 0555
  - name: tmp-data
    emptyDir: {}
  - name: prom-data
    emptyDir: {}
```

6. Grant security context constraint (SCC) `anyuid` to the required service accounts as shown in the following example.

```
$ oc adm policy add-scc-to-user anyuid -z default -n rag
$ oc adm policy add-scc-to-user anyuid -z rag-nv-ingest -n rag
$ oc adm policy add-scc-to-user anyuid -z rag-server -n rag
```

There are two ways to configure and install the Helm chart. Use an override values file to override the values in the Helm chart or make changes separately for every deployment or a sub chart.

Helm override values

Create a values file to override the values in the Helm chart for the following configuration areas:

- **GPU specific:** Depending on where the GPUs are located and which type of GPUs are used, nodeSelectors can be added for different components of the RAG. Additionally, add the NIM model profile for the specific GPU for the component nim-llm in the RAG pipeline.
- **Vector Store URL:** For the RAG server deployment, use the fully qualified domain name (FQDN) of the Milvus deployment for the environment variable APP_VECTORSTORE_URL.
- **Storage:** Enable persistence to cache the NIM model locally on the persistent volume claim (PVC). Specify the storage class name for the PVC or leave the field empty to use the default storage class. Update the size field to 200Gi. Depending on the LLM model, the size of the PVC may vary.

```
#----- overrides.yaml -----  
-----  
nodeSelector:  
  kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local  
envVars:  
  APP_VECTORSTORE_URL: "http://milvus.rag.svc.cluster.local:19530"  
  
frontend:  
  nodeSelector:  
    kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local  
  
ingestor-server:  
  nodeSelector:  
    kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local  
  envVars:  
    APP_VECTORSTORE_URL: "http://milvus.rag.svc.cluster.local:19530"  
  
nim-llm:  
  nodeSelector:  
    kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local  
  env:  
    - name: NIM_MODEL_PROFILE  
      value: "tensorrt_llm-rtx6000_blackwell_sv-fp8-tp1-pp1-throughput-2bb5:10de-  
d21d6986d29d8abf555f35c9a4c8146c4b10595d9e57e6efabd4a026efcc0c4a-1" # Provide correct profile  
name here  
  persistence:  
    enabled: true  
    storageClass: ""  
    size: 200Gi  
nvidia-nim-llama-32-nv-embedqa-1b-v2:  
  nodeSelector:  
    kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local  
  
nvidia-nim-llama-32-nv-rerankqa-1b-v2:  
  nodeSelector:  
    kubernetes.io/hostname: ocp-4-campus. ocp4.raplab.local  
  
nv-ingest:  
  nodeSelector:  
    kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local  
milvus:  
  standalone:
```

```

nodeSelector:
  kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local
minio:
  nodeSelector:
    kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local
etcd:
  nodeSelector:
    kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local
redis:
  master:
    nodeSelector:
      kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local
  replica:
    nodeSelector:
      kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local
paddleocr-nim:
  nodeSelector:
    kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local
nemoretriever-graphic-elements-v1:
  nodeSelector:
    kubernetes.io/hostname: ocp-0-campus.ocp4.raplab.local
nemoretriever-page-elements-v2:
  nodeSelector:
    kubernetes.io/hostname: ocp-0-campus.ocp4.raplab.local
nemoretriever-table-structure-v1:
  nodeSelector:
    kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local

```

Update Helm chart

To make the changes persistent in the chart, modify the files as described in the following steps.

1. To use GPU conservatively using time slicing, use the lighter LLM model. Replace the default LLM model with llama-3.1-nemotron-nano-8b-v1 in the file values.yaml.

```
$ sed -i '' 's/llama-3.3-nemotron-super-49b-v1.5/llama-3.1-nemotron-nano-8b-v1/g' values.yaml
```

2. Replace the tag for the model to be used.

```
$ sed -i '' 's/tag: "1.13.1"/tag: "1.8.4"/g' values.yaml
```

3. Using the following command, enable persistence for NIM LLM to cache the model locally. This configuration creates a PVC using the default storage class. Alternatively, edit the file directly and set `persistence.enabled: true`.

```
$ sed -i '' '/persistence:/,/^[^[:space:]]/ s/enabled:[[:space:]]*false/enabled: true/' charts/nim-llm/values.yaml
```

4. If the default StorageClass is not needed, update the field storageClass in all values.yaml files in the Helm chart.

```
sed -i '' 's/storageClass: ""/storageClass: "ocs-storagecluster-ceph-rbd"/g' $(grep -irl 'storageClass: ""' .)
```

5. If the plan is to use a different storage class for nim-llm, update the storageClass and size fields in file charts/nim-llm/values.yaml.

```
----- snippet of values.yaml -----
-----
persistence:
  enabled: true
  existingClaim: "" # if using existingClaim, run only one replica or use a
`ReadWriteMany` storage setup
  # Persistent Volume Storage Class
  # If defined, storageClassName: <storageClass>
  # If set to "-", storageClassName: "", which disables dynamic provisioning.
  # If undefined (the default) or set to null, no storageClassName spec is
  # set, choosing the default provisioner.
  storageClass: "ocs-storagecluster-ceph-rbd"
  accessMode: ReadWriteOnce # If using an NFS or similar setup, you can use
`ReadWriteMany`
  stsPersistentVolumeClaimRetentionPolicy:
    whenDeleted: Retain
    whenScaled: Retain
  size: 200Gi # size of claim in bytes (for example 8Gi)
```

6. Export the NGC API key.

```
$ export NGC_API_KEY=<key>
```

7. Install RAG Helm chart.

- a. If using the overrides file, install the Helm chart using the arguments shown in the following example.

```
$ helm upgrade --install rag ./ \
--username '$oauthtoken' \
--password "${NGC_API_KEY}" \
--set imagePullSecret.password=$NGC_API_KEY \
--set ngcApiSecret.password=$NGC_API_KEY \
--set nv-ingest.redis.image.repository=bitnamilegacy/redis \
--set nv-ingest.redis.image.tag=8.2.1-debian-12-r0
-f overrides.yaml -n rag
```

- b. If using the updated Helm chart, install the chart using the arguments shown in the following example.

```
$ helm upgrade --install rag ./ \
--username '$oauthtoken' \
--password "${NGC_API_KEY}" \
--set imagePullSecret.password=$NGC_API_KEY \
--set ngcApiSecret.password=$NGC_API_KEY \
--set nv-ingest.redis.image.repository=bitnamilegacy/redis \
--set nv-ingest.redis.image.tag=8.2.1-debian-12-r0 -n rag
```

Following is the expected output.

```

Release "rag" does not exist. Installing it now.
coalesce.go:237: warning: skipped value for etcd.extraVolumeMounts: Not a table.
coalesce.go:237: warning: skipped value for etcd.extraVolumes: Not a table.
I0106 23:00:32.999018 14427 warnings.go:110] "Warning:
spec.template.spec.containers[0].env[18]: hides previous definition of
\INGEST_LOG_LEVEL\", which may be dropped when using apply"
I0106 23:00:32.999091 14427 warnings.go:110] "Warning:
spec.template.spec.containers[0].env[47]: hides previous definition of
\VLM_CAPTION_ENDPOINT\", which may be dropped when using apply"
I0106 23:00:32.999101 14427 warnings.go:110] "Warning:
spec.template.spec.containers[0].env[65]: hides previous definition of
\OTEL_EXPORTER_OTLP_ENDPOINT\", which may be dropped when using apply"
NAME: rag
LAST DEPLOYED: Tue Jan 6 22:59:26 2026
NAMESPACE: rag
STATUS: deployed
REVISION: 1

```

8. Verify that all pods are in the Running and Ready state.

```

$ oc get pod -n rag
NAME                                READY STATUS RESTARTS   AGE
ingestor-server-65b858cf4d-rk52z    1/1   Running 1 (176m ago) 176m
milvus-standalone-7588f6787f-wlc96  1/1   Running 3 (175m ago) 176m
nv-ingest-ocr-75bc9c7bdd-85pgr      1/1   Running 0         176m
rag-etcd-0                           1/1   Running 0         176m
rag-frontend-75dbf7f8d9-c85sf       1/1   Running 0         176m
rag-minio-5bb67c8d9f-xvpn6          1/1   Running 0         176m
rag-nemoretriever-graphic-elements-v1-68bf49c49d-f4scf  1/1   Running 0         176m
rag-nemoretriever-page-elements-v2-86655669f4-tkfvp    1/1   Running 0         176m
rag-nemoretriever-table-structure-v1-6c96bb8b66-68fxb  1/1   Running 0         176m
rag-nim-llm-0                        1/1   Running 0         176m
rag-nv-ingest-7c6c84cb5c-kscpn      1/1   Running 0         176m
rag-nvidia-nim-llama-32-nv-embedqa-1b-v2-569467f68b-n42xt 1/1   Running 0         176m
rag-nvidia-nim-llama-32-nv-rerankqa-1b-v2-7f8b6b6f9c-ln6d8 1/1   Running 0         176m
rag-redis-master-0                  1/1   Running 0         176m
rag-redis-replicas-0                1/1   Running 0         176m
rag-server-667fbfc449-ldvcv         1/1   Running 3 (173m ago) 176m

```

Following are the services of every component in the RAG pipeline.

```

$ oc get svc -n rag
NAME                                TYPE           CLUSTER-IP     EXTERNAL-IP  PORT(S)          AGE
ingestor-server                     ClusterIP      172.30.214.108 <none>       8082/TCP        177m
milvus                               ClusterIP      172.30.175.232 <none>       19530/TCP,9091/TCP 177m
nemoretriever-embedding-ms          ClusterIP      172.30.20.169  <none>       8000/TCP        177m
nemoretriever-graphic-elements-v1  ClusterIP      172.30.173.112 <none>       8000/TCP,8001/TCP 177m
nemoretriever-page-elements-v2      ClusterIP      172.30.113.62  <none>       8000/TCP,8001/TCP 177m
nemoretriever-ranking-ms            ClusterIP      172.30.229.78  <none>       8000/TCP        177m
nemoretriever-table-structure-v1    ClusterIP      172.30.161.117 <none>       8000/TCP,8001/TCP 177m
nim-llm                             ClusterIP      172.30.210.135 <none>       8000/TCP        177m
nim-llm-sts                         ClusterIP      None            <none>       8000/TCP        177m
nv-ingest-ocr                       ClusterIP      172.30.146.224 <none>       8000/TCP,8001/TCP 177m
rag-etcd                            ClusterIP      172.30.8.232   <none>       2379/TCP,2380/TCP 177m
rag-etcd-headless                   ClusterIP      None            <none>       2379/TCP,2380/TCP 177m
rag-frontend                        NodePort       172.30.218.71  <none>       3000:30761/TCP   177m
rag-minio                           ClusterIP      172.30.68.238  <none>       9000/TCP        177m
rag-nv-ingest                       ClusterIP      172.30.51.62   <none>       7670/TCP        177m

```

rag-redis-headless	ClusterIP	None	<none>	6379/TCP	177m
rag-redis-master	ClusterIP	172.30.8.150	<none>	6379/TCP	177m
rag-redis-replicas	ClusterIP	172.30.90.159	<none>	6379/TCP	177m
rag-server	ClusterIP	172.30.195.38	<none>	8081/TCP	177m

9. Run the following command to port-forward the RAG UI service to the local machine.

```
$ oc port-forward -n rag service/rag-frontend 3000:3000 --address 0.0.0.0
```

Appendix C: NVIDIA RAG prompt interface

This section describes how to interact with the NVIDIA RAG Prompt Interface after the RAG Blueprint has been successfully deployed. It demonstrates how users can ingest documents, create collections, and issue queries to validate retrieval and grounded response generation. This interface also serves as a reference point for verifying that the RAG pipeline is operational before deploying and using the NVIDIA AI-Q in subsequent appendices.

The following steps illustrate how to use the NVIDIA RAG Prompt Interface to ingest documents and issue queries against the deployed RAG pipeline:

1. Access the RAG UI at <http://localhost:3000>. Click **+ New Collection** to create a collection of documents.

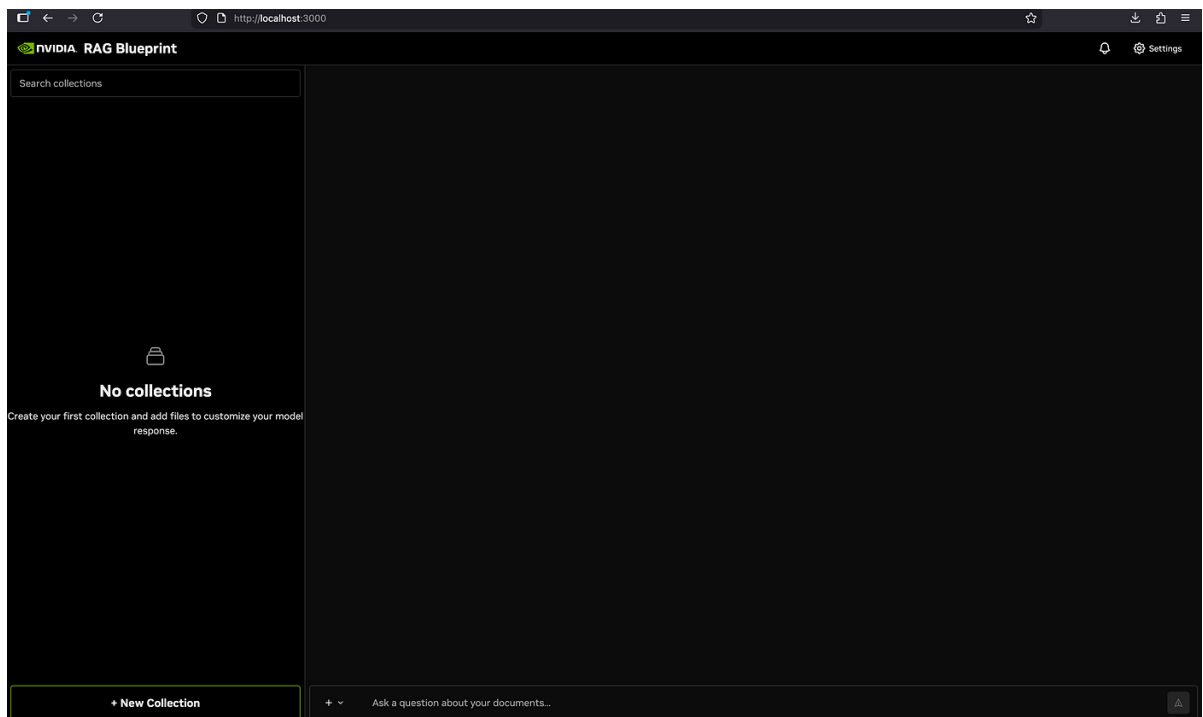


Figure 5: RAG Blueprint UI

- In the left panel, enter the relevant name for the collection under the field **Collection Name**. In the right panel, upload one or more documents to ingest in the RAG pipeline. In the example shown in the following figure, a single PDF is selected for uploading. Click **Create Collection** to create a collection and upload a document in the collection.

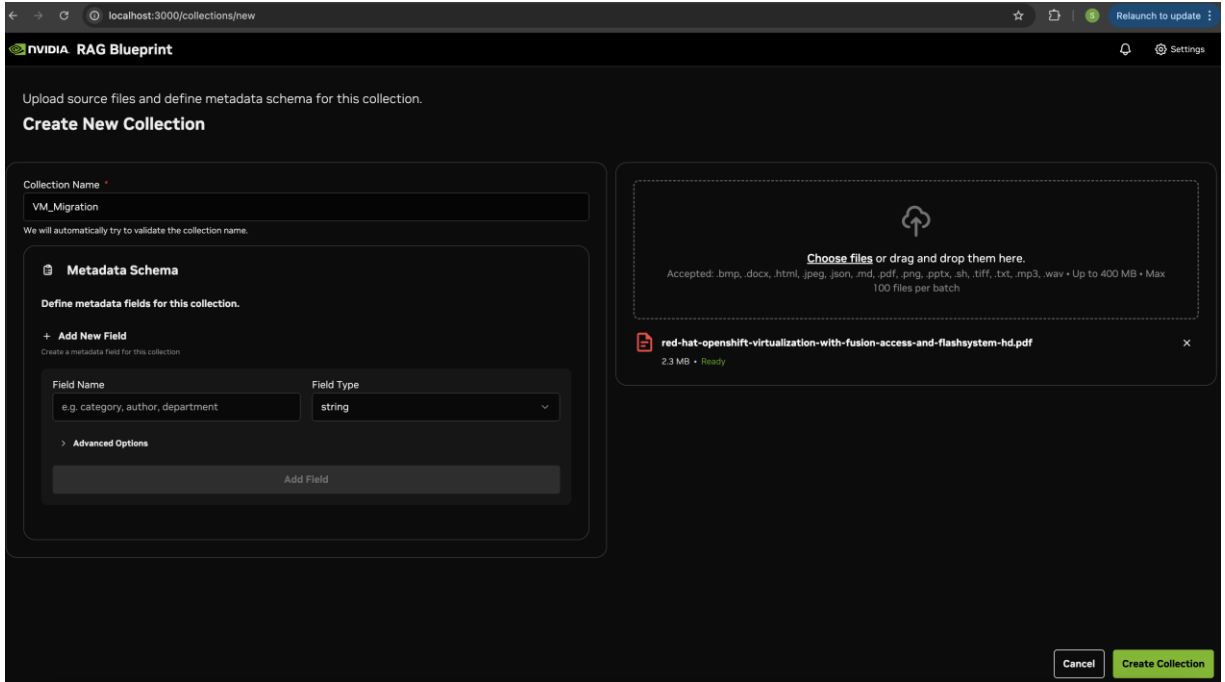


Figure 6: Create collection of documents

- Observe the notification window displaying the progress of the ingestion.

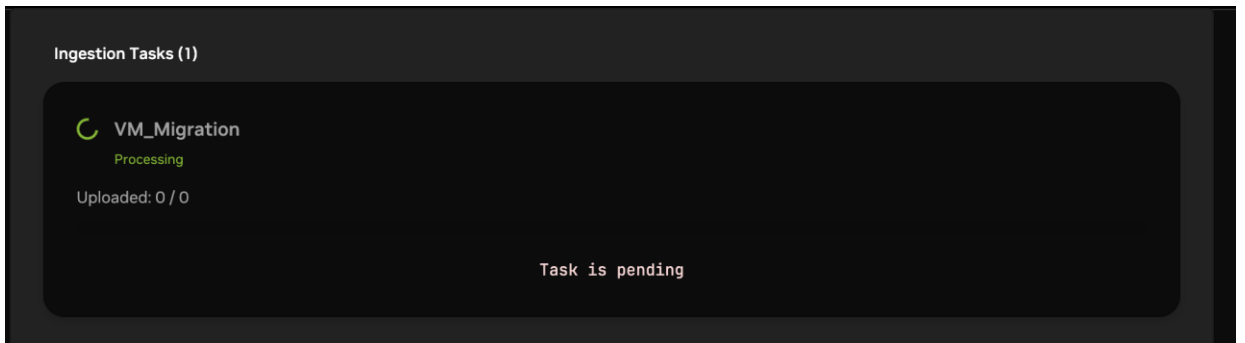


Figure 7: Ingestion

- Verify that, upon successful ingestion, the status is shown as **Completed**.

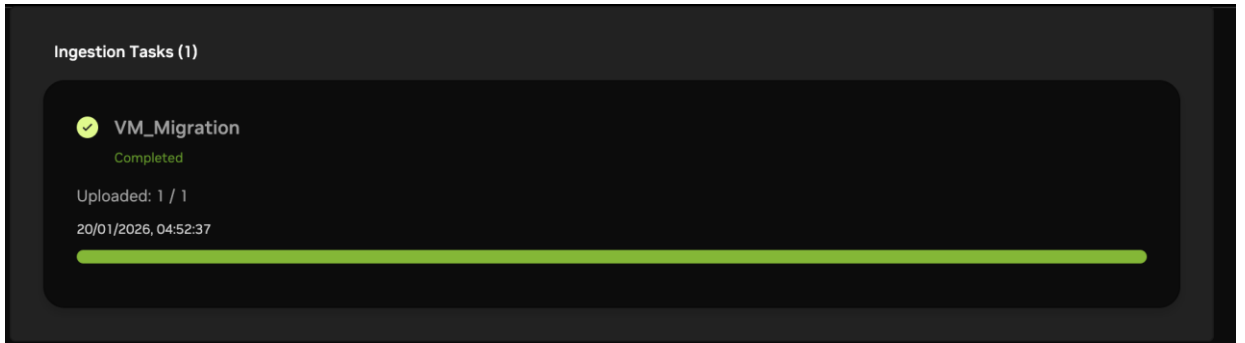


Figure 8: Successful ingestion

- Run prompts or questions after the documents are ingested to retrieve information from the RAG pipeline. The following figure shows an example of prompts to run queries against the ingested document.

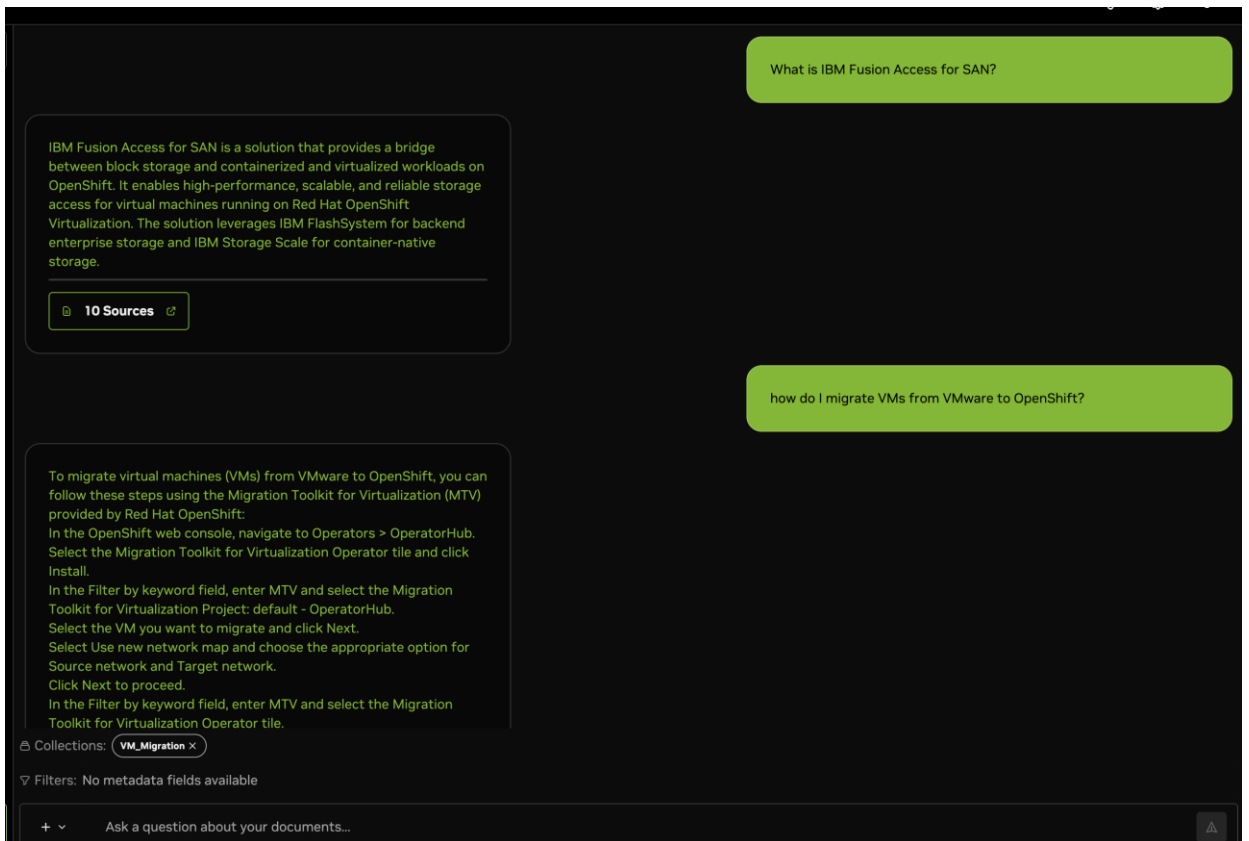


Figure 9: Prompts and answers

6. Click the Sources link to see which documents were used to find the information.

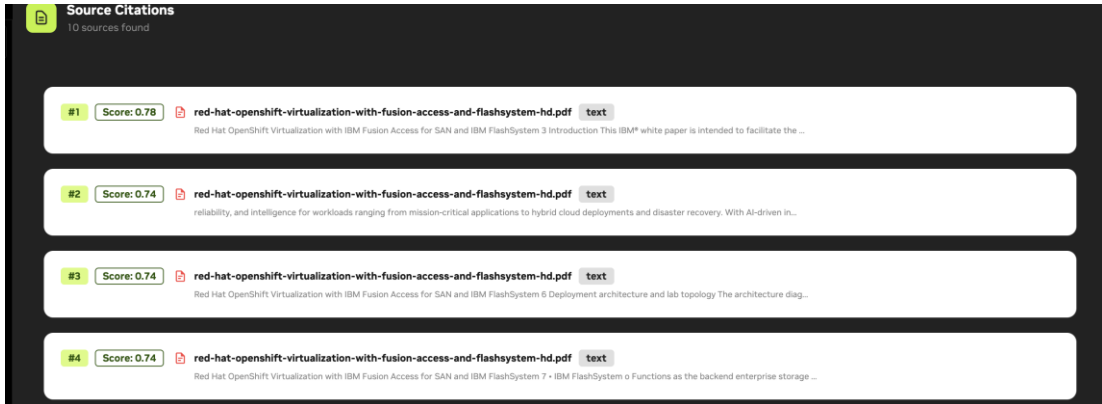


Figure 10: Source citations

Appendix D: NVIDIA AI-Q deployment

This section describes the steps required to deploy the NVIDIA AI-Q Blueprint on IBM Fusion HCI. This deployment builds on an existing NVIDIA RAG Blueprint installation and introduces the agentic orchestration, reasoning, and report-generation capabilities that extend the RAG pipeline. The appendix covers Helm-based installation, configuration customization, and validation of the AI-Q components, and assumes that the RAG deployment described in Appendix B has been successfully completed and verified.

The following steps outline the process for configuring and deploying the NVIDIA AI-Q Blueprint on IBM Fusion HCI:

1. Download the chart (AI Q bundle references RAG components).

```
$ wget https://helm.ngc.nvidia.com/nvidia/blueprint/charts/nvidia-blueprint-rag-v2.3.0.tgz
```

2. Extract the downloaded file.

```
$ tar xvzf nvidia-blueprint-rag-v2.3.0.tgz
```

3. Create a namespace where all the AI-Q components will be running.

```
$ oc create namespace aiq
```

4. Grant security context constraint (SCC) anyuid to the required service accounts as shown in the following example.

```
$ oc adm policy add-scc-to-user anyuid -z default -n aiq
```

There are two ways to configure and install the Helm chart. Use an override values file to override the values in the Helm chart or make changes separately for every deployment or a subchart.

Override values

Create a values file to override the values in the Helm chart. The steps that follow are for RTX6000 Blackwell GPUs that are available on the node **ocp-4-campus.ocp4.raplab.local**.

- **GPU specific:** Depending on where the GPUs are located and which types of GPUs are used, nodeSelectors can be added for AI-Q. Additionally, add the NIM model profile for the specific GPU for nim-llm.
- **Storage:** Enable persistence to cache the NIM model locally on the PVC. Specify the storage class name to be used for the PVC or keep the field empty to use the default storage class. Update the size field to use 200Gi. Depending on the LLM model, the size of the PVC may vary.

```
#----- overrides.yaml -----  
-----  
  
nim-llm:  
  nodeSelector:  
    kubernetes.io/hostname: ocp-4-campus.ocp4.raplab.local  
  env:  
    - name: NIM_MODEL_PROFILE  
      value: "tensorrt_llm-rtx6000_blackwell_sv-fp8-tp1-pp1-throughput-2bb5:10de-  
d21d6986d29d8abf555f35c9a4c8146c4b10595d9e57e6efabd4a026efcc0c4a-1" # Provide correct profile  
name here  
  persistence:  
    enabled: true  
    storageClass: ""  
    size: 200Gi
```

Update Helm chart

To make the changes persistent in the chart, modify the files from the chart directory as described in the following steps.

1. The default llama-3.3-70b-instruct requires approximately 120 GB of VRAM and, to use GPUs conservatively, choose a lighter model. Replace it with llama-3.2-3b-instruct.

```
$ sed -i '' 's/llama-3.3-70b-instruct/llama-3.2-3b-instruct/g' values.yaml
```

2. Replace the default Nemotron model in the file values.yaml of AI-Q. If the default model was replaced in the RAG Helm chart, replace it here as well in the AI-Q chart.

```
$ sed -i '' 's/llama-3.3-nemotron-super-49b-v1.5/llama-3.1-nemotron-nano-8b-v1/g'  
values.yaml
```

3. Update the GPU count to 1 in the file values.yaml.

```
$ sed -i 's/nvidia.com/gpu: 2/nvidia.com/gpu: 1/g' values.yaml
```

4. Enable persistence for NIM LLM to cache the model locally. This creates a PVC using the default storage class. Alternatively, edit the file directly and set `persistence.enabled: true`.

```
$ sed -i '' '/persistence:\/,\/^[^[:space:]]\/ s/enabled:[[:space:]]*false/enabled: true/'
charts/nim-llm/values.yaml
```

5. If not planning to use the default storage class, update the `storageClass` field in all `values.yaml` files in the Helm chart.

```
sed -i '' 's/storageClass: ""/storageClass: "ocs-storagecluster-ceph-rbd"/g' $(grep -irl
'storageClass: ""' .)
```

6. If the plan is to use some other storage class for `nim-llm` model, update the `storageClass` and `size` fields in file `charts/nim-llm/values.yaml`.

```
----- snippet of values.yaml -----
-----
persistence:
  enabled: true
  existingClaim: "" # if using existingClaim, run only one replica or use a
`ReadWriteMany` storage setup
  # Persistent Volume Storage Class
  # If defined, storageClassName: <storageClass>
  # If set to "-", storageClassName: "", which disables dynamic provisioning.
  # If undefined (the default) or set to null, no storageClassName spec is
  # set, choosing the default provisioner.
  storageClass: "ocs-storagecluster-ceph-rbd"
  accessMode: ReadWriteOnce # If using an NFS or similar setup, you can use
`ReadWriteMany`
  stsPersistentVolumeClaimRetentionPolicy:
    whenDeleted: Retain
    whenScaled: Retain
  size: 200Gi # size of claim in bytes (for example 8Gi)
```

7. Export the `NGC_API` key.

```
$ export NGC_API_KEY=<key>
```

8. Export the `TAVILY_API` key if planning to use web search in AI-Q.

```
$ export TAVILY_API_KEY=<key>
```

9. Install AI-Q Helm chart.

- a. If using the overrides file, install the Helm chart using the arguments shown in the following figure.

```
$ helm upgrade --install aiq-aira . -n aiq \
--username='${oauthToken}' \
--password=${NGC_API_KEY} \
--set imagePullSecret.password=${NGC_API_KEY} \
--set ngcApiSecret.password=${NGC_API_KEY} \
--set tavilyApiSecret.password=${TAVILY_API_KEY} -n aiq
```

- b. If updating the Helm chart, install the Helm chart using the arguments shown in the following figure.

```
$ helm upgrade --install aiq-aira . -n aiq \
--username='$oauthtoken' \
--password=$NGC_API_KEY \
--set imagePullSecret.password=$NGC_API_KEY \
--set ngcApiSecret.password=$NGC_API_KEY \
--set tavilyApiSecret.password=$TAVILY_API_KEY
-f overrides.yaml -n aiq
```

The following is the expected output.

```
Release "aiq-aira" does not exist. Installing it now.
I0115 00:19:25.116057 39489 warnings.go:110] "Warning:
spec.template.spec.containers[0].env[12]: hides previous definition of
\"INSTRUCT_API_KEY\", which may be dropped when using apply"
NAME: aiq-aira
LAST DEPLOYED: Thu Jan 15 00:19:15 2026
NAMESPACE: aiq
STATUS: deployed
REVISION: 1

$ oc get pod -n aiq
NAME                                READY STATUS   RESTARTS   AGE
aiq-aira-aira-backend-855d9c4bb5-wr6xq 1/1   Running    0           46h
aiq-aira-aira-frontend-59d9c897f6-24s1t 1/1   Running    0           46h
aiq-aira-nim-llm-0                       1/1   Running    0           129m
aiq-aira-phoenix-78fd7584b7-rthct       1/1   Running    0           46h
```

10. Verify that all pods are in the Running and Ready state.

11. List the services. Note down the node port of the frontend service.

```
$ oc get svc -n aiq
NAME                                TYPE           CLUSTER-IP      EXTERNAL-IP  PORT(S)                                AGE
aiq-aira-aira-backend               ClusterIP      172.30.9.93     <none>       3838/TCP                               46h
aiq-aira-aira-frontend               NodePort       172.30.178.51   <none>       3000:30080/TCP                         46h
aiq-aira-phoenix                     ClusterIP      172.30.216.188 <none>       6006/TCP,4317/TCP                     46h
instruct-llm                         ClusterIP      172.30.65.143   <none>       8000/TCP                                46h
instruct-llm-sts                     ClusterIP      None             <none>       8000/TCP                                46h
```

Appendix E: NVIDIA AI-Q experience dashboard

The section illustrates the NVIDIA AI-Q experience through its web-based dashboard. It demonstrates how users interact with the deployed AI-Q system to define research topics, select document sources, review agent-generated plans, and generate structured research reports. This appendix provides a practical view of the end-to-end agentic research workflow, showing how retrieval, reasoning, and synthesis are combined to deliver grounded and traceable outputs in an enterprise environment.

The following figures illustrate the core user workflows and interactions available in the NVIDIA AI-Q dashboard.

The AI-Q web UI provides a browser-based interface for interacting with the research assistant. Users access this interface through the worker node IP and service port, enabling natural language research queries and report generation.

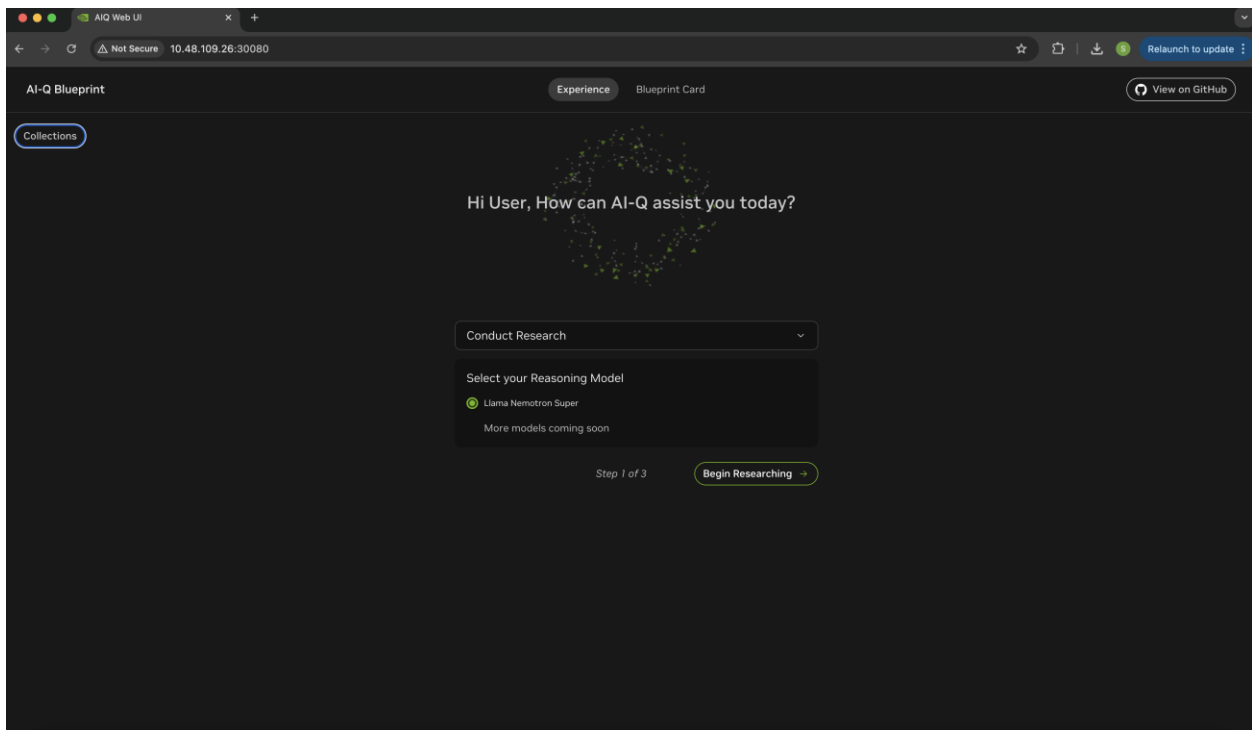


Figure 11: AI-Q Web UI

Figure 12 shows the AI Q interface where users select an existing document collection and define the report topic and output structure. The left panel lists available document collections, while the content area allows users to specify the research subject and guide the structure of the generated report.

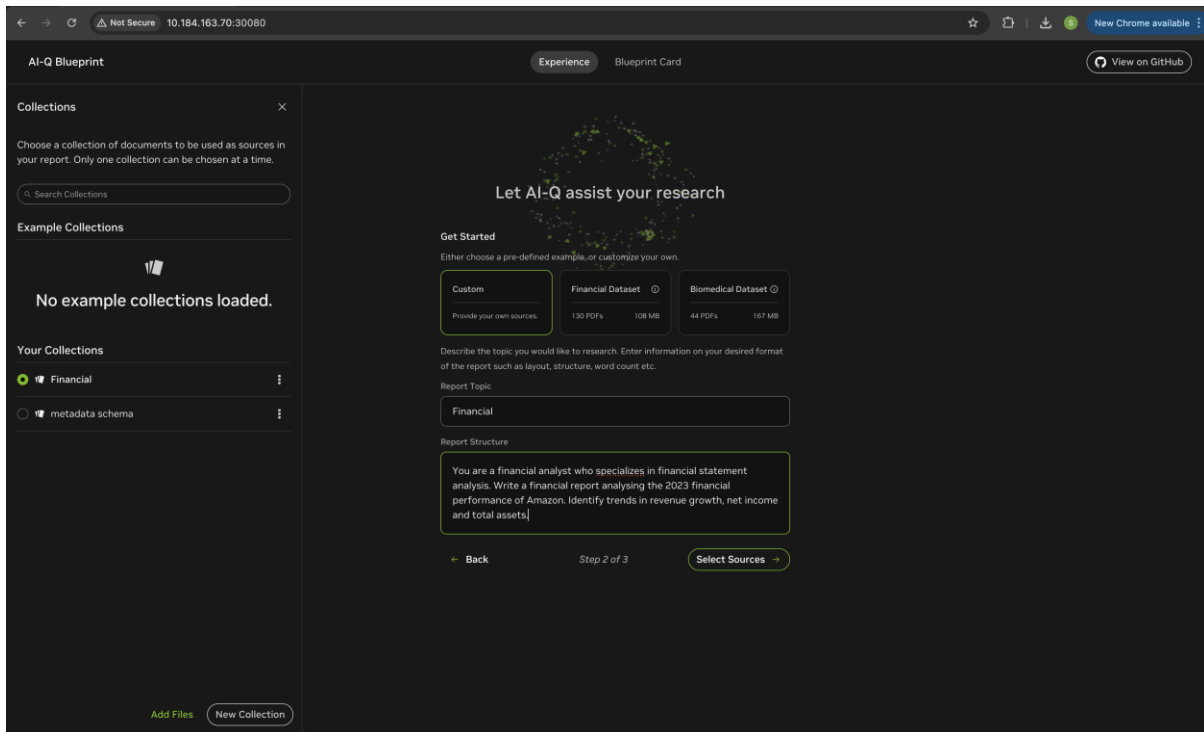


Figure 12: Selecting the topic Research Content and Defining the Report

The AI Q assistant enables users to attach source material for the research task. In this step, users can select existing document collections and optionally enable web search to supplement enterprise data sources before starting report generation.

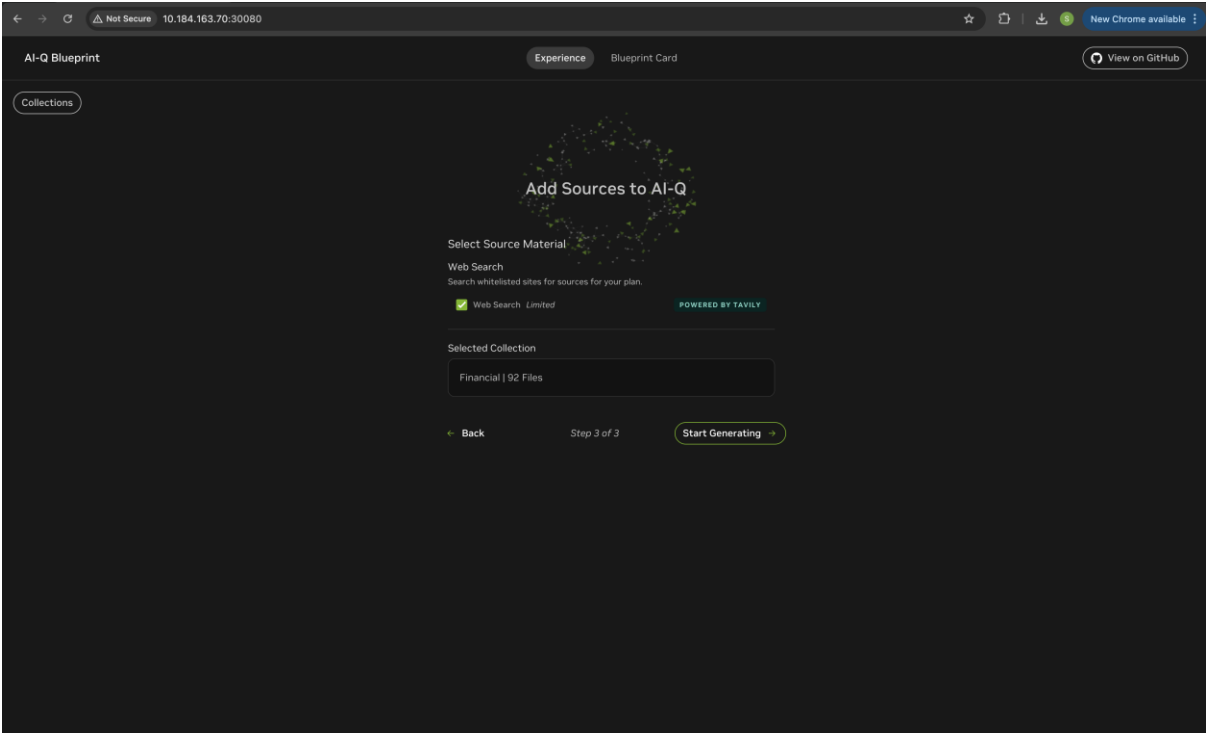


Figure 13: Adding the session Source Material to the AI-Q Research

This screen illustrates the AI-Q agent thinking phase.

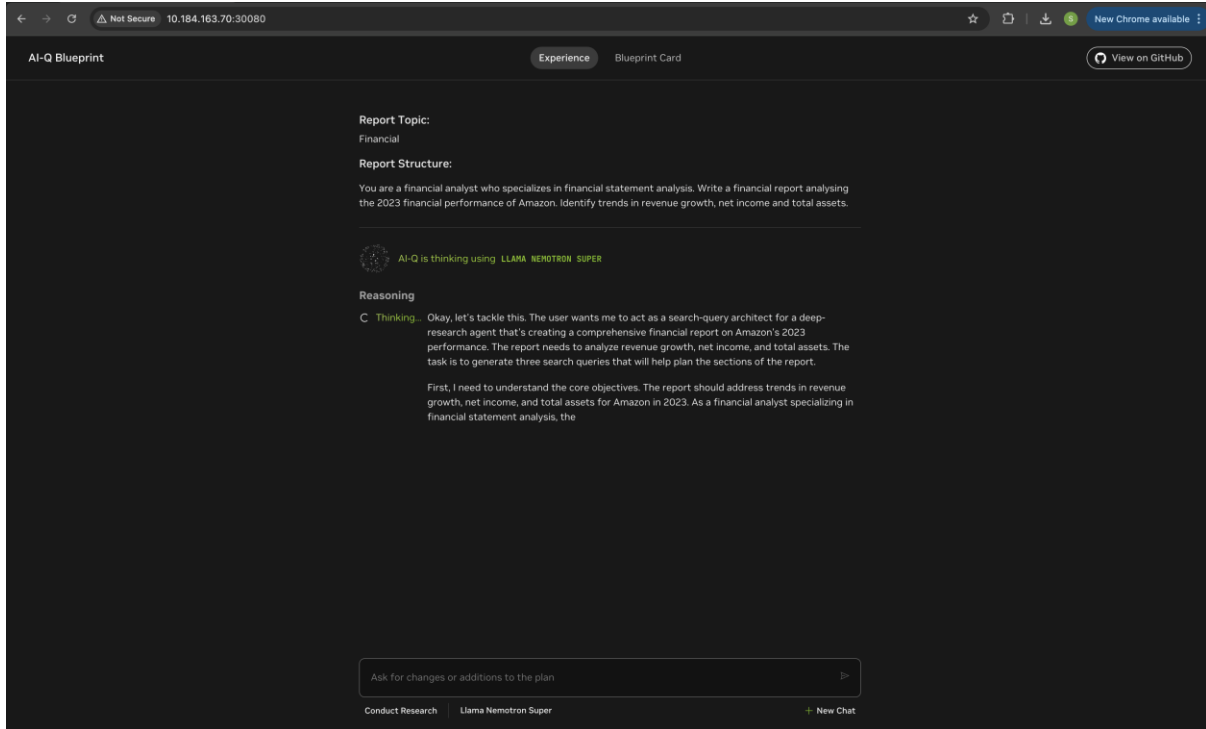


Figure 14: AI-Q Thinking phase

This screen illustrates the AI Q agent planning and reasoning phase, where the assistant analyzes the research request, decomposes it into logical steps, and presents a structured plan. Users can review the proposed approach and choose to proceed or refine the plan before execution.

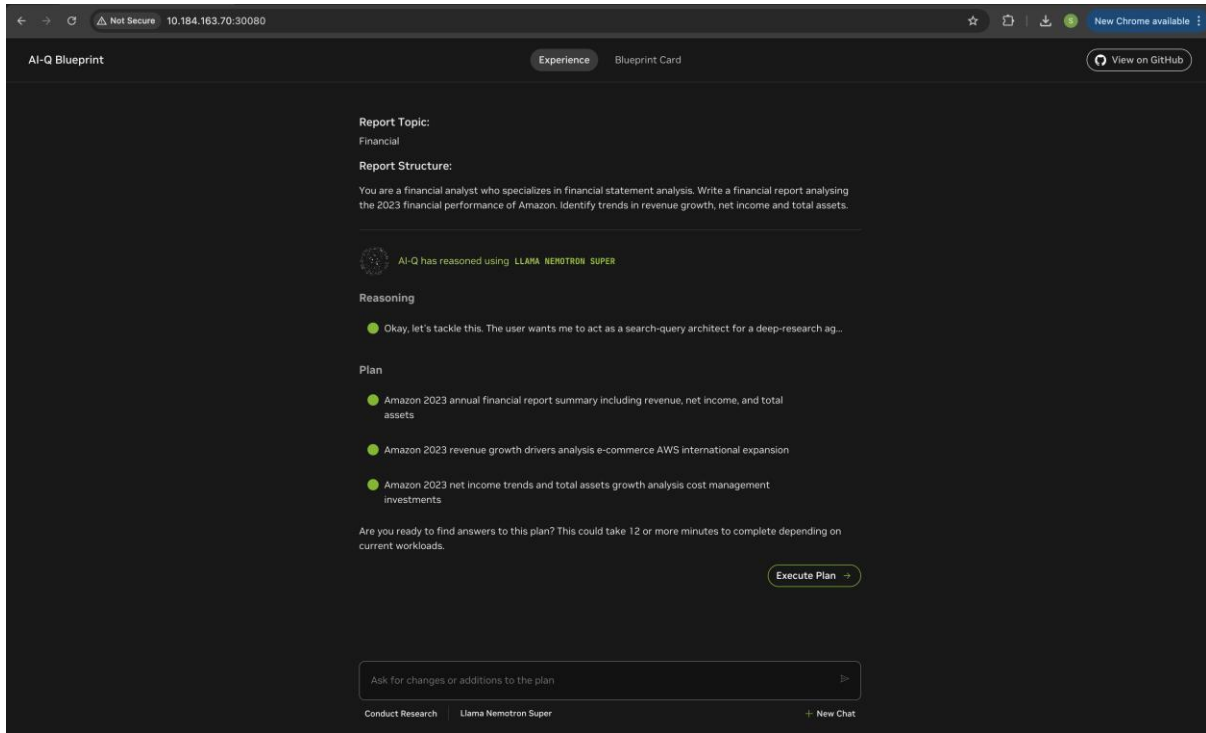


Figure 15: AI-Q research planning and Reasoning Phase

After the research plan is executed, AI Q begins gathering relevant information from connected data sources using the underlying RAG pipeline. Figure 16 shows the system in the data retrieval phase, indicating that documents are being searched, ranked, and prepared for synthesis.

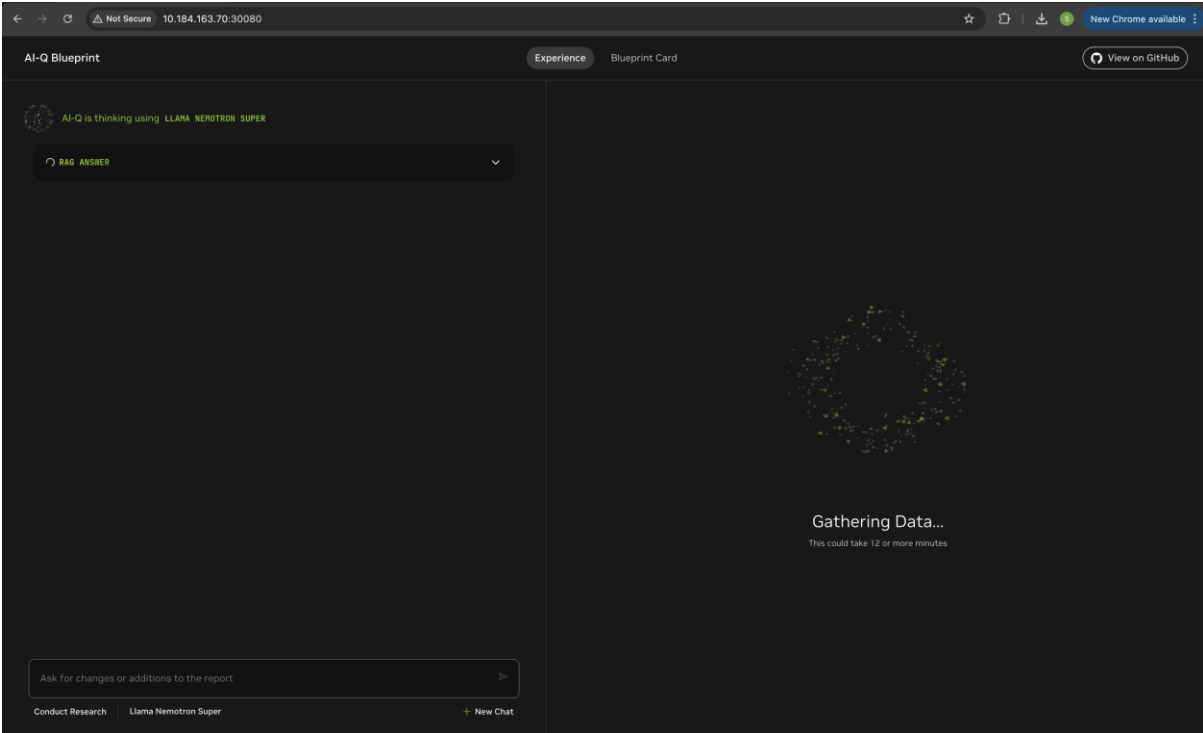


Figure 16: Data collection and retrieval in progress

AI Q provides detailed responses backed by retrieved content. Figure 17 displays the content retrieved through RAG search.

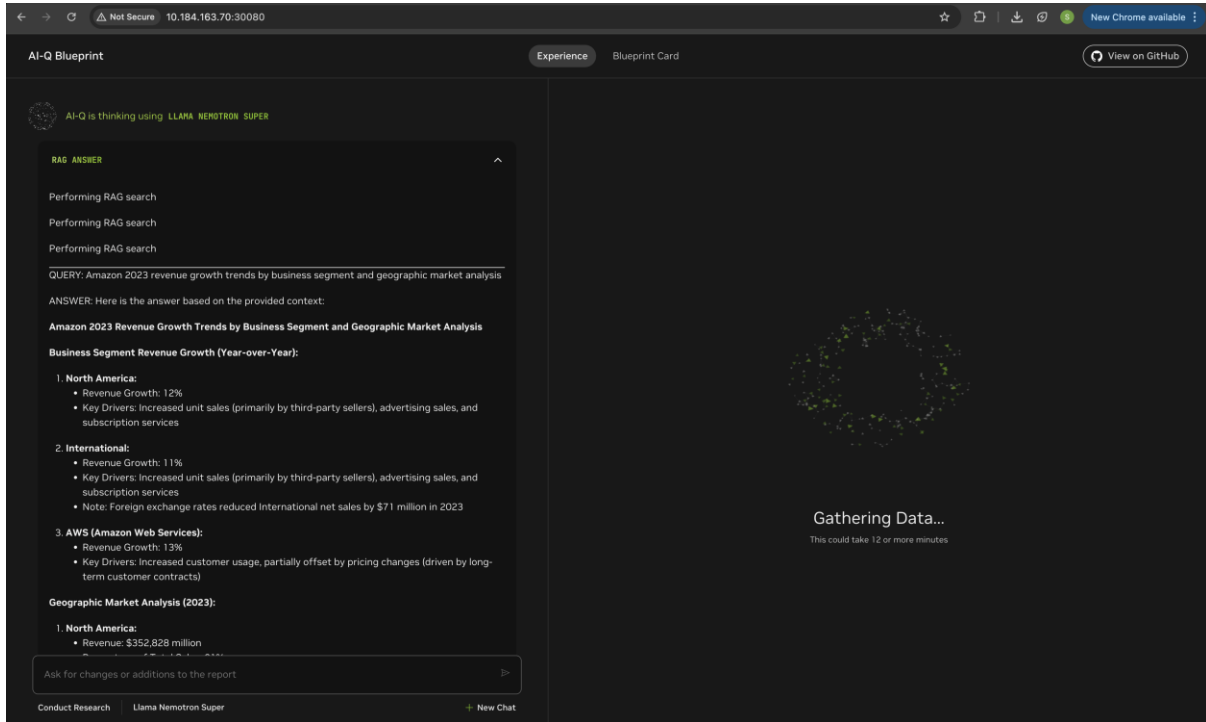


Figure 17: RAG search

Figure 18 displays the generated research report produced by AI Q. The output is organized into clear sections, combining retrieved enterprise data with model reasoning to deliver a comprehensive and grounded report. This view highlights individual answers supporting explanations, and associated source references, demonstrating how the assistant ensures traceability and transparency in its responses. Users can review, refine, or export the results.

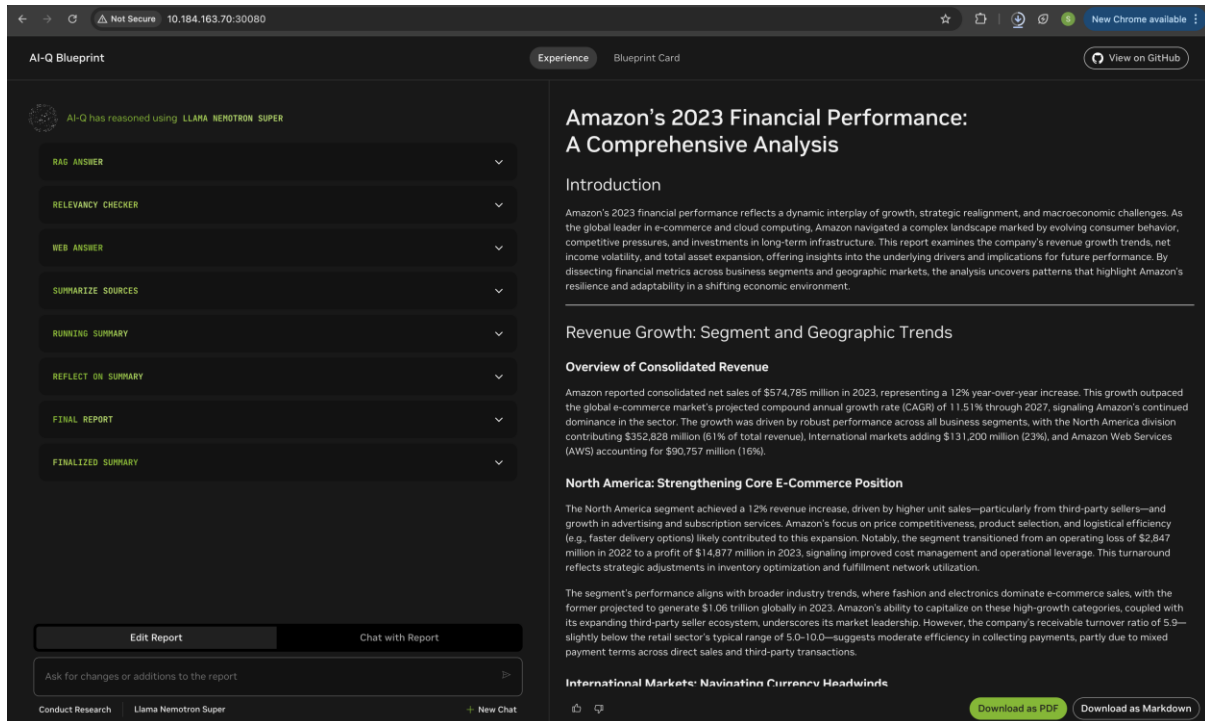


Figure 18: Generated research report with structured sections

© Copyright IBM Corporation 2026

IBM Corporation
One Orchard Road
Armonk, NY 10504

Produced in the
United States of America
April 2026

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, in the United States and/or other countries. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on ibm.com/trademark.

Red Hat and OpenShift are registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

NIM and NeMo are trademarks of NVIDIA or its subsidiaries in the United States and other countries.

This document is current as of the initial date of publication and may be changed by IBM at any time. Not all offerings are available in every country in which IBM operates. All client examples cited or described are presented as illustrations of the manner in which some clients have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual client configurations and conditions. Generally expected results cannot be provided as each client's results will depend entirely on the client's systems and services ordered. It is the user's responsibility to evaluate and verify the operation of any other products or programs with IBM products and programs. THE INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING WITHOUT ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND ANY WARRANTY OR CONDITION OF NON-INFRINGEMENT. IBM products are warranted according to the terms and conditions of the agreements under which they are provided.