RHOAI + RAG Architecture Document

Document Status

Author:	Akram Ben Aissi
Reviewer:	Varsha Prasad Narsing Christian Zaccaria Mark Campbell Francisco Arceo
Date:	2025-06-10
Document Status:	Draft

Document Review

Reviewer	Starting date	End date	Status
Varsha Prasad Narsing	2025-06-10		In Progress

Table of contents

This document describes the software architecture used to integrate Red Hat OpenShift AI (RHOAI) with LlamaStack to develop a Retrieval-Augmented Generation (RAG) based solution for OpenShift. This integration enables a robust and scalable system for handling complex queries and generating informed responses by combining OpenShift's enterprise AI platform with LlamaStack's flexible LLM capabilities.

Introduction to LlamaStack

LlamaStack is Meta's service-oriented **framework** that simplifies large language model (LLM) application development through a unified set of REST APIs. The platform provides essential LLM capabilities including:

- Inference
- Safety
- Agentic workflows
- Vector operations
- Evaluation
- Telemetry

All of these functions are made accessible through consistent **REST** interfaces, regardless of the underlying provider. Its key strength lies in **provider flexibility**—developers can seamlessly switch between backends (e.g., vLLM for inference, Milvus for vector storage) without changing application logic.

LlamaStack is available in multiple pre-configured **distributions**, from cloud to mobile environments. This abstraction simplifies infrastructure management while preserving the developer's choice of implementation.

RHOAI + RAG Overview

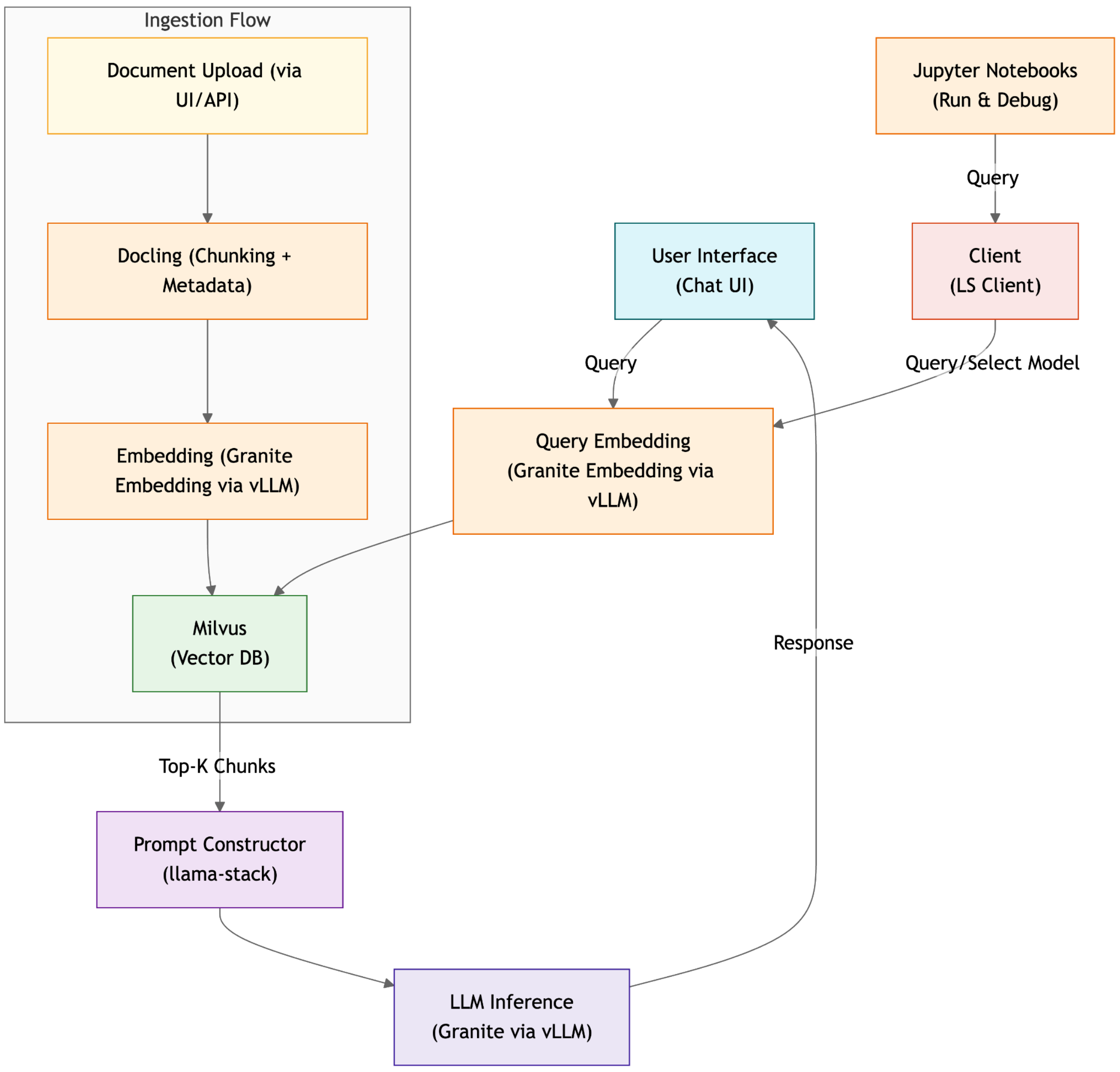
The goal of this architecture is to leverage both OpenShift AI and LlamaStack to deliver an efficient and enterprise-grade RAG solution under a custom **Red Hat LlamaStack Distribution.**

- **RHOAI** offers the platform capabilities for managing and deploying ML/AI models.
- **LlamaStack** provides composable LLM services and orchestrates retrieval-based workflows.

Together, they provide a foundation for integrating internal and external knowledge sources with generative AI in a secure, scalable, and manageable way.

High-Level Architecture

The high level architecture of our solution can be explained by the following diagram:


Ingestion Flow
Document Upload (via UI/API)
Docling (Chunking + Metadata)
Embedding (Granite Embedding via vLLM)
Milvus (Vector DB)
Jupyter Notebooks (Run & Debug)
Query
User Interface (Chat UI)
Client (LS Client)
Query
Query/Select Model
Query Embedding (Granite Embedding via vLLM)
Response
Top-K Chunks
Prompt Constructor (llama-stack)
LLM Inference (Granite via vLLM)

In our llama-stack RAG workflow, an OpenShift user uploads a document through a chat-style UI (or an API), which is then processed by a KubeFlow pipeline that runs Docling to extract and chunk the text that it contains. The supported document formats are typically doc and pdf.

The chunks are embedded using a model like Granite, served via vLLM and KServe, and the resulting vectors are stored in Milvus. When the user later asks a question, it is embedded the same way and used to search Milvus for the most relevant chunks.

Then, these chunks, along with the user's query, are assembled into a prompt and optionally passed through a Kubeflow pipeline for tasks like re-ranking or logging.

Finally, the prompt is sent to the LLM, which generates a response that is returned to the user in the UI, creating a seamless, document-aware chat experience.

Key Components

The architecture is composed of foundational elements from both OpenShift AI (RHOAI) and LlamaStack. Each component plays a distinct role in the end-to-end Retrieval-Augmented Generation (RAG) pipeline, ensuring modularity, scalability, and extensibility.

RHOAI Integration Layer

OpenShift AI provides the infrastructure backbone for model serving, development tooling, and persistent storage.

- **Model Serving with vLLM on KServe**: OpenShift leverages KServe to deploy and manage large language models efficiently. vLLM serves as the model inference engine, optimized for high-throughput GPU usage. KServe handles autoscaling, traffic routing, and lifecycle management of model endpoints, ensuring robust performance under varying loads.

- **Jupyter Notebooks as User-Facing Tools**: Pre-configured Jupyter notebooks allow users to interact with the deployed models directly. These notebooks are used for prompt experimentation, model tuning, testing retrieval workflows, and visualizing results. They serve both as a development workspace and as a learning tool for new users.

- **Milvus Vector Store with Persistent Storage**: The system includes an inline Milvus vector database that indexes document embeddings for fast semantic search. In the RHOAI environment, this is backed by a Persistent Volume Claim (PVC), allowing the index to survive restarts and pod migrations. The design supports both scalability and high availability of vector retrieval operations.

LlamaStack Components

LlamaStack introduces the orchestration logic and LLM-related services required for RAG.

- **RAG Engine**: This core engine coordinates the entire retrieval-to-generation workflow. It receives user queries, performs embedding-based vector retrieval, manages prompt construction, and interacts with the LLM backend for final response generation.

- **Granite Model for Embeddings and Generation**: A fine-tuned Granite model is used for both vector embedding and response generation. Hosted via vLLM, it ensures consistency between the way documents are indexed and queried. Its integration within LlamaStack provides flexibility in switching models or versions.

- **Docling Document Processor**: Docling is used in the document ingestion pipeline. It processes uploaded documents, parses them into structured formats, and performs chunking. This preprocessing is critical to ensure that the vector representation aligns semantically with downstream queries.

Kubeflow Pipelines for Ingestion Automation

Kubeflow Pipelines automate the multi-step process of populating the vector store. The pipeline includes steps for document upload (triggered manually or programmatically), document parsing and chunking (via Docling), embedding generation (via Granite model on vLLM), and vector storage in Milvus. This modular and automated approach ensures consistency, repeatability, and auditability of the document ingestion process.

Workflow

1. **User Query Submission**: The user inputs a question or prompt through a chatbot UI or notebook interface.

2. **Retrieval:** LlamaStack RAG Engine receives the user query and initiates the retrieval phase. This involves interpreting the user's question and formulating a semantic search request. The engine uses the embedded representation of the query to look for relevant documents or document fragments.

3. **Contextual Retrieval**: Milvus vector store receives the semantic query from the RAG engine. Milvus performs a similarity search against its indexed document embeddings. These embeddings were previously generated and stored during the document ingestion pipeline. The store returns the top-k closest matches—text fragments that are most similar in meaning to the user's query.

4. **Response Generation**: The query and context are combined and sent to the deployed Granite model via vLLM on KServe.

5. **Response Delivery**:
The generated answer is sent back to the user, completing the RAG loop.

Step	Action	Component
1	Query Submission	User Interface
2	Retrieval	LlamaStack Engine
3	Context Retrieval	Vector Database
4	Generation	OpenShift AI Model Deployment
5	Response Delivery	User Interface

Data FlowIntegration Details

The integration of RHOAI with LlamaStack offers a comprehensive foundation for building enterprise-grade, retrieval-augmented AI services. This section expands on the architectural benefits and operational considerations that make this solution robust, scalable, and adaptable to various use cases within OpenShift environments.

Scalability

The system leverages OpenShift's dynamic orchestration capabilities on different aspects:

- KServe's model autoscaling allows adaptation to varying workloads. As query volume grows or new models are deployed, resources can be scaled horizontally with minimal manual intervention. This ensures responsive performance even under high demand.
- KubeFlow Pipeline parallelisation and dynamic resources allocation allows scalable data ingestion
- Milvus can be installed either inline (pod collococated) or remote. When support for remote Milvus will be available, we can fully benefit from scalability and availability features. In the meanwhile, Pod's configurable resources allocation allows horizontal scaling.

Modularity and Flexibility

Each component in the architecture is loosely coupled and can be upgraded or replaced independently. For example, the Granite model could be swapped for another LLM with minimal changes to the overall pipeline. This modularity allows rapid innovation and customization without disrupting the core system.

Developer Experience

Jupyter Notebooks offer a familiar and flexible environment for developers and data scientists to experiment, prototype, and interact with the deployed models. Combined with LlamaStack's REST APIs, the platform supports a seamless development-to-deployment workflow.

Retrieval-Augmented Precision

The use of a RAG engine grounded in vector-based retrieval reduces hallucination and ensures that model responses are anchored in factual, domain-specific content. This approach improves trustworthiness and context awareness, which are critical in enterprise applications.

Security and Governance

Security is enforced through OpenShift's OAuth provider and RBAC policies, ensuring that users only access authorized data and services. Multi-tenant isolation and auditability further support governance, compliance, and secure collaboration across organizational boundaries.

Operational Transparency

With integrated observability from both OpenShift and LlamaStack, the solution provides full visibility into system health, performance metrics, and user behavior. This enables proactive monitoring, fast troubleshooting, and continuous optimization of the RAG pipeline.

Additional Integrations

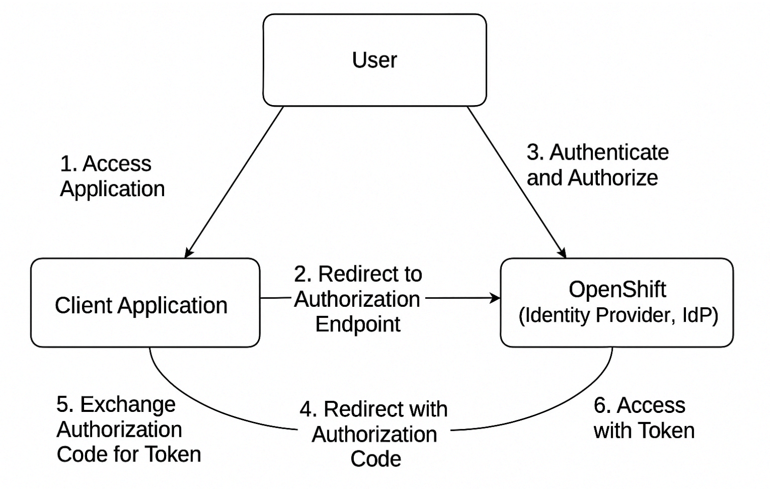
To further enhance the capabilities of the RHOAI and LlamaStack RAG solution, a number of additional integrations are envisioned. These focus on expanding user experience, improving observability, enforcing enterprise security standards, and enabling dynamic and composable AI services.

Conversational UI with PatternFly

The RAG solution will integrate a conversational user interface built with PatternFly, Red Hat's enterprise design system. This web interface will allow users to submit questions, view previous responses, and upload documents for real-time ingestion. Features like session history, bookmarks, and shareable conversation links will enhance usability, while feedback options will help continuously improve model behavior.

Authentication and Authorization with OpenShift OAuth

User access will be governed through OpenShift's integrated OAuth provider. When users access the interface or backend APIs, they will be redirected to OpenShift's OAuth flow for authentication.


User
1. Access Application
3. Authenticate and Authorize
Client Application
2. Redirect to Authorization Endpoint
OpenShift (Identity Provider, IdP)
5. Exchange Authorization Code for Token
4. Redirect with Authorization Code
6. Access with Token

The platform will support enterprise identity providers, allowing users to log in with their corporate credentials. Once authenticated, role-based access control (RBAC) will ensure users can only access the models, documents, and operations permitted by their roles. This approach supports secure, multi-tenant environments and allows for detailed usage tracking and enforcement of resource quotas.

Observability and Monitoring

Observability will be handled through the integration of OpenShift's native monitoring stack with metrics and logs exposed by LlamaStack services. Administrators will have access to dashboards displaying key performance indicators such as query latency, embedding times, model response quality, and system resource utilization. Alerts can be configured to detect anomalies or degradation in service levels, allowing for proactive intervention.

Multi-Model Routing

To support a flexible and extensible model serving layer, the system will implement multi-model routing. Based on request metadata—such as user identity, workload type, or performance constraints—queries can be routed to different model backends. For example, some requests may go to a locally deployed Granite model, while others may use Llama3 hosted externally. The routing logic will be centrally managed and policy-driven.

Conclusion

The integration of Red Hat OpenShift AI with LlamaStack presents a robust and modular architecture for enterprise-grade Retrieval-Augmented Generation (RAG). By combining OpenShift's scalable infrastructure with LlamaStack's composable LLM services, the solution addresses the need for context-aware, reliable, and secure AI in modern organizations.

Each layer of the architecture—from vectorized document ingestion to real-time inference and retrieval—is designed with modularity, observability, and security in mind. The use of OpenShift-native tools ensures compatibility with enterprise scalability, monitoring and governance workflows, while LlamaStack simplifies the application layer for developers.

This foundation is suitable for building intelligent assistants, domain-specific chatbots, and knowledge-extraction tools that are both performant and trustworthy. With planned enhancements such as multi-model routing, federated retrieval, and a PatternFly-powered UI, the architecture is well-positioned for long-term evolution and production deployment.

Appendices

This section contains the links to the demos performed across the development of the RHOAI+RAG solution and that exhibits every feature described in this document.

1. Christian Zaccaria built an example of interacting with a Llama Stack RAG application from a RHOAI Workbench using a GPU-accelerated vLLM (Llama 3.2) model via KServe.
2. Akram Ben Aissi showed an MVP for our "One Click" basic RAG deployment to empower customers to get started using Llama Stack, Milvus, and vLLM
3. Alex Creasy demoed the MVP ChatBot UI
    a. Notice the rendering code rendering!
4. Varsha Prasad Narsing showed her upstream work to enable keyword and vector search with RAG