



AI Factory

Release 1.3

EnterpriseDB Corporation

Oct 27, 2025

Contents

1	Where AI Factory runs	2
1.1	Prerequisites	3
1.2	Why Sovereign AI?	3
1.3	What You Can Build	3
1.4	Choose your path	4
1.5	Who Is It For, and How to Get Started?	4
1.6	Core Capabilities	4
1.6.1	Gen AI Workloads	4
1.6.2	Asset Library and Model Serving	5
1.6.3	Pipelines and Vector Engine	5
1.7	EDB PG AI and Analytics	5
1.8	Learn and Build	5
1.9	Build Your EDB PG AI Solution	5
2	AI Factory	7
2.1	AI Factory Hub Map	7
2.1.1	Start here	7
2.1.2	Build with AI Factory	7
2.1.3	Hybrid Manager	8
2.1.4	Where it runs and prerequisites	8
2.1.5	References	8
2.2	AI Factory Learning Center	8
2.2.1	Quickstarts	8
2.2.2	Conceptual Explanations	9
2.2.3	How-to Guides	9
2.2.4	Learning Paths	9
2.2.5	Solutions	9
2.2.6	Use Cases	9
	Getting Started - AI Factory 101	10
	Learning Path Modules	11
	Continue Your Learning Journey	13
	Building Production-Ready AI Features - AI Factory 201	13
	Modules by Focus Area	14
	Next steps	15
	Related learning resources	15
	Advanced AI Factory Usage and Extensibility - AI Factory 301	16

	Modules by Focus Area	16
	Next steps	17
	Related learning resources	18
	AI Factory Explained	18
	AI Factory How-To Guides	19
	Learning Paths	20
	Learning Paths	20
	Recommended Training Courses	22
	Where to next?	22
	Industry Solutions with AI Factory	22
	AI Factory Use Cases	25
2.3	Concepts	29
2.4	Concepts	30
	2.4.1 EDB’s vision for AI with Postgres and Hybrid Manager	30
2.5	Concepts	30
2.6	Concepts	30
2.7	Concepts	30
2.8	Concepts	31
	2.8.1 Summary	31
	AI Factory vs. “Generic AI”	31
	Fundamental AI Concepts	34
	AI Factory terminology	39
	Sovereign AI Explained	43
	Security and governance in AI Factory	45
3	Usage	48
3.1	AI Factory Models	48
3.2	AI Factory Models	48
	3.2.1 System Overview	48
	Core Components	48
	3.2.2 Strategic Benefits	49
	Sovereign AI Implementation	49
	Operational Consistency	49
	Scalable Resource Management	49
	3.2.3 Architecture Integration	49
	Hybrid Manager Dependencies	49
	3.2.4 Operational Workflows	49
	Model Lifecycle Management	49
	Governance Framework	50
	3.2.5 Learning Pathways	50
	Foundation Understanding	50
	Practical Implementation	50
	Reference Materials	51
	3.2.6 Use Case Applications	51
	Enterprise RAG Systems	51
	Private LLM Deployment	51
	Multi-Modal AI Applications	51
	Hybrid Model Architectures	51
	3.2.7 Implementation Considerations	51
	Infrastructure Requirements	51
	Network Architecture	51
	Security Framework	51
	Operational Complexity	52
3.3	Gen AI Builder	133

3.4	Gen AI Builder	133
3.4.1	Platform Architecture	134
	Core Infrastructure Integration	134
3.4.2	Architectural Components	134
	Knowledge Base Framework	134
	Retrieval Systems	134
	Assistant Framework	135
3.4.3	Implementation Patterns	135
	Enterprise Knowledge Systems	135
	Retrieval-Augmented Generation Applications	135
	Conversational AI Platforms	136
3.4.4	Development Framework	136
	Knowledge Base Development	136
	Assistant Configuration	136
	API Development	137
3.4.5	Integration Architecture	137
	EDB Postgres AI Ecosystem	137
	External System Integration	137
3.4.6	Operational Considerations	138
	Performance Characteristics	138
	Security Framework	138
	Governance Integration	138
3.4.7	Getting Started	138
	Prerequisites	138
	Initial Implementation	139
	Learning Resources	139
3.5	AI Factory Pipelines	261
3.5.1	Why Pipelines?	261
3.5.2	What You Can Do	262
3.5.3	How It Works	262
	Preparers	262
	Knowledge Bases	262
	Auto-Processing	262
3.5.4	Who Should Use Pipelines?	263
3.5.5	When to Use Pipelines	263
3.5.6	Configuration values explained	263
3.5.7	Learn More	263
	Start here	263
	Get started	267
	Core concepts	274
	Storage and access	366
	Pipelines reference	374
	EDB Postgres AI - AI Accelerator release notes	402
	Legal	412
3.6	Vector Engine (PGvector)	426
3.6.1	Vector Engine	426
3.6.2	Why Vector Search in Postgres?	427
3.6.3	Who Should Use Vector Engine?	427
3.6.4	What Does It Provide?	427
3.6.5	How Vector Engine Works	428
3.6.6	When Should You Use Vector Engine?	428
3.6.7	Example Use Cases	428
3.6.8	Integration with AI Accelerator Pipelines and Gen AI Builder	428
3.6.9	Next Steps	429

3.7	AI Factory Release Notes	429
3.7.1	AI Factory 1.3 release notes	429
	Release Highlights	429
	Deployment Options	429
	New Features & Enhancements	430
	Observability, Monitoring & Insights	431
	Sustainability: ESG Reporting (Preview)	431
	Platform Improvements	431
	Resources	432

EDB Postgres® AI provides a scalable, modular platform to build and operationalize AI-powered applications and intelligent data products — starting from **your data**, in **your databases**, running in **your data center or cloud environments**. **EDB PG AI is designed for Hybrid Manager™ first**. Most capabilities are delivered and managed inside [Hybrid Manager's AI Factory](#), where you get a unified control plane, governance, and observability. You can also run selected components outside Hybrid Manager — notably Postgres extensions and AI DB (AIDB) — when supported by licensing.

At its core, EDB Postgres® AI (EDB PG AI) is designed to enable **Sovereign AI**: allowing you to build intelligence on top of your trusted Postgres data, governed within your enterprise boundaries. Your models, your pipelines, your vector indexes — all managed and controlled by you.

With EDB PG AI, you can:

- Build and run **Gen AI agents** and conversational experiences powered by your enterprise knowledge and sovereign Postgres data in **Hybrid Manager**
 - Perform **vector search** and **retrieval-augmented generation (RAG)** over structured and unstructured sources
 - Create **document intelligence pipelines** that ingest, process, and classify data from diverse sources
 - Deploy scalable [Model Serving Reference Manual](#), using GPU acceleration and enterprise observability
 - Extend Postgres with AI-driven query patterns and **vector-native capabilities**
 - Seamlessly combine AI and analytics across Postgres, EDB Lakehouse, and other EDB-integrated platforms
-

CHAPTER 1

Where AI Factory runs

EDB Postgres® AI provides a scalable, modular platform to build and operationalize AI-powered applications and intelligent data products — starting from **your data**, in **your databases**, running in **your data center or cloud environments**. **EDB PG AI is designed for Hybrid Manager™ first**. Most capabilities are delivered and managed inside [Hybrid Manager’s AI Factory](#), where you get a unified control plane, governance, and observability. You can also run selected components outside Hybrid Manager — notably Postgres extensions and AI DB (AIDB) — when supported by licensing.

At its core, EDB Postgres® AI (EDB PG AI) is designed to enable **Sovereign AI**: allowing you to build intelligence on top of your trusted Postgres data, governed within your enterprise boundaries. Your models, your pipelines, your vector indexes — all managed and controlled by you.

With EDB PG AI, you can:

- Build and run **Gen AI agents** and conversational experiences powered by your enterprise knowledge and sovereign Postgres data in **Hybrid Manager**
- Perform **vector search** and **retrieval-augmented generation (RAG)** over structured and unstructured sources
- Create **document intelligence pipelines** that ingest, process, and classify data from diverse sources
- Deploy scalable [Model Serving Reference Manual](#), using GPU acceleration and enterprise observability
- Extend Postgres with AI-driven query patterns and **vector-native capabilities**
- Seamlessly combine AI and analytics across Postgres, EDB Lakehouse, and other EDB-integrated platforms

Choose the right operating model for your teams and environments:

- Hybrid Manager (recommended hub): Use [AI Factory in Hybrid Manager](#) to provision, govern, and observe AI workloads across Postgres, pipelines, vector search, model serving, and Gen AI.
- Outside Hybrid Manager (selected components): Run Postgres-focused capabilities on compatible platforms, such as the [Vector Engine \(PGvector\)](#), Pipelines and data preparation, or the AI DB (AIDB) feature set where available.

When you operate outside Hybrid Manager, you own cluster lifecycle, security, and observability. Validate compatibility, performance, and licensing for your target platform before deployment.

1.1 Prerequisites

What you need depends on where you run:

- Using Hybrid Manager: Access to a Hybrid Manager instance with AI Factory enabled and the entitlements required for Gen AI, model serving, and vector search. See [AI Factory in Hybrid Manager](#).
- Running components outside Hybrid Manager: A supported Postgres environment for extensions and AIDB, and the necessary infrastructure for pipelines and model serving as applicable.

For precise component support (outside Hybrid Manager), minimum versions, and licensing boundaries, coordinate with your EDB representative and your platform owner.

1.2 Why Sovereign AI?

EDB PG AI enables you to create **Sovereign AI solutions** that respect your data ownership, security, and governance requirements:

- Operate on **your data** — in your Postgres databases and EDB analytics lakes
- Train and serve **your models**, under your control — no opaque third-party APIs required
- Deploy **where you need** — in Hybrid Manager, on-prem data centers, or in your cloud tenancy
- Integrate AI, analytics, and operational data into unified, governed architectures
- Audit, monitor, and control the full lifecycle of AI workloads — from ingestion to inference

By combining Postgres, modern AI pipelines, and scalable model serving in a fully governed architecture, EDB PG AI helps you unlock AI innovation **without compromising trust or compliance**.

1.3 What You Can Build

EDB PG AI is designed to help you create intelligent applications and systems **on top of your trusted data** — with Postgres at the center.

You can build:

- **Enterprise knowledge assistants** powered by Gen AI and Postgres-native **vector search**
- **AI-powered search portals** that integrate structured, semi-structured, and unstructured data
- **Document processing pipelines** with **Pipelines**, OCR, summarization, classification, and extraction
- **Conversational applications** for both internal users and external customers, built with [Knowledge Bases Reference Manual](#)
- **AI-powered data apps** exposing [Model Serving Reference Manual](#) for search, recommendations, NLP, and vision
- **Intelligent automation** embedded in business systems and workflows — driven by your data

- **RAG architectures** combining Postgres, Lakehouse, and Gen AI to deliver trusted answers

See our *AI Factory Use Cases* and *Industry Solutions with AI Factory* for detailed examples and solution patterns.

1.4 Choose your path

Use these entry points based on your goal (Diataxis):

- Concepts and architecture: *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- How-to guides: *Gen AI How-To Guides* and *How-To Deploy AI Models from Model Library*
- Tutorials and learning: *Learning Paths* — 101, 201, and 301
- Reference: Gen AI reference , *Model Serving Reference Manual* , *Vector Engine (PGvector)* , and Pipelines and data preparation

1.5 Who Is It For, and How to Get Started?

EDB PG AI serves **data-driven organizations** that want to build AI applications and systems **on top of their core Postgres and analytics estate**:

- **Developers** building Gen AI apps and intelligent data experiences
- **Data engineers** managing **Pipelines** , **vector search** , and hybrid data preparation
- **AI engineers** operationalizing models with *Model Serving Reference Manual* within governed Postgres-integrated environments
- **Data architects** designing AI + analytics solutions at scale
- **IT leaders and security teams** seeking enterprise-grade control and auditability

Getting started is easy:

- Explore our *Learning Paths* to build foundational skills
- Start building **knowledge bases** and **assistants** with *Knowledge Bases Reference Manual* and *Assistants Reference Manual*
- Deploy your first *Model Serving Reference Manual* and integrate with Postgres queries
- Build **pipelines** and **vector search** to power intelligent applications on trusted data

1.6 Core Capabilities

Explore each capability area of EDB PG AI:

1.6.1 Gen AI Workloads

- *Gen AI Workloads* — Overview of capabilities
- *Assistants Reference Manual* — Build assistants, tools, structures, and rulesets
- *Knowledge Bases Reference Manual* — Manage knowledge bases and data lakes

1.6.2 Asset Library and Model Serving

- *Model Library Reference Manual* — Manage and govern models and AI assets
- *Model Serving Reference Manual* — Deploy and scale model inference services

1.6.3 Pipelines and Vector Engine

- Pipeline Capabilities — Build pipelines for document intelligence and vectorization
- *Vector Engine (PGvector)* — Integrate vector search and AI-driven query patterns into Postgres

1.7 EDB PG AI and Analytics

EDB PG AI is deeply integrated with the **EDB Analytics ecosystem** :

- Combine structured + unstructured data for AI-powered search and insights
- Build Lakehouse-backed RAG pipelines that feed Gen AI agents
- Store and query vector embeddings natively alongside Postgres and Lakehouse data
- Create hybrid AI + analytics solutions that leverage Postgres extensions, analytics engines, and AI serving — all governed within EDB PG AI

This enables truly **data-driven AI** — where the best of your operational, transactional, and analytical data powers your AI systems.

1.8 Learn and Build

Go deeper with the following resources:

- *AI Factory Explained* — Understand the architecture and components of EDB PG AI
- *Gen AI How-To Guides* — Practical guides for building EDB PG AI solutions

1.9 Build Your EDB PG AI Solution

Start today with one or more of these steps:

- Build your first *Assistants Reference Manual* using Agent Studio
- Deploy a *Knowledge Bases Reference Manual* with Gen AI Builder and integrate with RAG
- Launch your first *Model Serving Reference Manual*
- Create **Data Pipelines** for document intelligence
- Add **Vector Search** to Postgres

Explore *AI Factory Use Cases* and *Industry Solutions with AI Factory* for end-to-end patterns and architecture guidance.

EDB PG AI gives you a powerful, modular foundation to build AI-powered solutions **on your data, in your datacenter or cloud, governed by you.**

Start building today.

2.1 AI Factory Hub Map

Use this page to jump directly to what you need. Each tile links to the right place in the hub or the Hybrid Manager spoke. For prerequisites and where components run, see the AI Factory landing page.

2.1.1 Start here

- Concepts: Understand the architecture and components
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *AI Factory terminology*
- Learning: Follow guided paths and use cases
- *Learning Paths*
- *AI Factory Use Cases and Industry Solutions with AI Factory*

2.1.2 Build with AI Factory

- Gen AI (assistants, rulesets, tools, threads)
- Gen AI hub
- *Customer Service Assistant Implementation Guide*
- Pipelines (RAG, knowledge bases, auto□processing)
- Pipelines and data preparation
- *Knowledge Bases*
- Vector Engine (vector search in Postgres) - *Vector Engine (PGvector)*
- Model (serving + library)
- *Model Serving Reference Manual*

- *Model Library Reference Manual*

2.1.3 Hybrid Manager

- AI Factory in Hybrid Manager: Governed operations, observability, GPUs
- Spoke root
- Gen AI in HM
- *Pipelines models reference*
- GPU management in HM

2.1.4 Where it runs and prerequisites

- Where AI Factory runs (Hybrid Manager first; selected components outside HM) - See landing page: Where it runs
- Prerequisites (entitlements, GPUs where applicable) - See landing page: Prerequisites

2.1.5 References

- Gen AI reference - SDK, tools, drivers
- Model reference - *Model Serving Reference Manual*
- Pipelines reference - Capabilities, PGFS, models

2.2 AI Factory Learning Center

Welcome to the **AI Factory Learning Center** — your central starting point for building with **EDB Postgres® AI** (EDB PG AI).

EDB PG AI enables **Sovereign AI** — using your data, in your databases and data center, to power intelligent applications and data products. The platform builds directly on Postgres with integrated vector search, Pipelines, Gen AI Builder, Knowledge Bases, and Model Serving.

Use this Learning Center to:

- Understand key concepts and architecture
- Follow step-by-step how-to guides
- Progress through structured learning paths
- Complete practical tutorials
- Explore solution patterns and real-world use cases

2.2.1 Quickstarts

Get productive fast with role-based guides:

- *Create an Assistant in Gen AI Builder* — Build a customer service assistant
- Data Engineer quickstart — Stand up a RAG pipeline with Pipelines + Vector Engine
- *How-To Setup GPU resources* — Serve a private model with KServe and GPUs

2.2.2 Conceptual Explanations

Start here to understand the architecture, core components, and how EDB PG AI works:

- *AI Factory Explained*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Knowledge Bases Reference Manual*
- *Model Serving Concepts*
- *Sovereign AI Explained* — Why Sovereign AI matters and how EDB PG AI enables trusted, governed AI on your data and infrastructure.

2.2.3 How-to Guides

Step-by-step guides for building with EDB PG AI:

- *Gen AI How-To Guides*
- *Create a Knowledge Base in Gen AI Builder*
- *Configure the Data Lake in Gen AI Builder*
- *How-To Create an InferenceService for NVIDIA NIM Container*

2.2.4 Learning Paths

Structured learning paths — start here if you're new or want a guided journey:

- *Learning Paths*
- *Getting Started - AI Factory 101*
- *Building Production-Ready AI Features - AI Factory 201*
- *Advanced AI Factory Usage and Extensibility - AI Factory 301*

2.2.5 Solutions

Reference architectures and solution patterns for enterprise needs:

- *Industry Solutions with AI Factory*
- Example solution patterns:
 - Gen AI-powered search portal
 - Knowledge assistant with Retrieval Augmented Generation (RAG)
 - Intelligent document processing pipeline

2.2.6 Use Cases

Explore real-world applications of EDB PG AI:

- *AI Factory Use Cases*
- Enterprise knowledge assistant
- Intelligent automation in workflows
- AI-powered recommendations and search

Tip: If you're new to EDB PG AI, start with *Getting Started - AI Factory 101*.

You can also dive deeper into core capabilities:

- Pipelines and data preparation
 - *Gen AI Builder*
 - *Model Serving Reference Manual*
 - *Vector Engine (PGvector)*
-

This Learning Center evolves continuously — check back often for new guides, tutorials, and learning resources.

Start building intelligent applications and data products with **EDB Postgres® AI** today.

Getting Started - AI Factory 101

Take your AI Factory skills to the next level — master complex Assistants, advanced data pipelines, and enterprise-grade model serving.

Who Should Take This Path

This intermediate learning path is designed for:

- **Application Developers** integrating AI features into production systems
- **Data Engineers** building robust pipelines for AI-driven Knowledge Bases
- **AI & Platform Engineers** implementing model serving and observability
- **Enterprise Teams** deploying Sovereign AI workloads at scale

Prerequisites: Completion of *Getting Started - AI Factory 101* or equivalent experience with basic AI Factory concepts.

Learning Outcomes

Upon completing this path, you will master:

OK Advanced AI Orchestration — Build multi-step Assistants with complex personas, persistent memory, and tool integrations

OK Enterprise Data Pipelines — Design Hybrid Knowledge Bases with multi-source RAG, metadata filtering, and hybrid search

OK Production Model Serving — Deploy and optimize GPU-powered models using KServe with performance tuning

OK Operational Excellence — Implement comprehensive observability, monitoring, and governance for AI-driven applications

Estimated Total Time: 2-3 hours | **Difficulty:** Intermediate

Learning Path Modules

1. Advanced Assistant & Structure Design

- Master sophisticated AI orchestration patterns*

What You'll Learn:

- Design Assistants with nuanced personas and contextual memory
- Build reliable multi-step Structures with error handling
- Integrate external Tools and data sources securely

Time Investment: 30-45 minutes

Modules:

1. **** *Assistants Reference Manual* **** — Deep dive into assistant architecture, persona design, and memory management
 2. **** *Structures Reference Manual* **** — Master workflow orchestration with branching logic and validation patterns
 3. **** *Create an Assistant in Gen AI Builder* **** — Hands-on: Build a production assistant with integrated tools and knowledge bases
 4. **** *Create a Structure in Gen AI Builder* **** — Hands-on: Implement complex workflows with conditional logic and error recovery
 5. **** *Create a Tool in Gen AI Builder* **** — Hands-on: Wrap internal APIs as governed, secure AI tools
-

2. Data Engineering & Hybrid Knowledge Bases

- Build enterprise-grade data pipelines for AI*

What You'll Learn:

- Architect Hybrid Knowledge Bases for optimal performance and cost
- Implement advanced RAG patterns with multiple data sources
- Configure intelligent search with metadata filtering and ranking

Time Investment: 30-45 minutes

Modules:

1. **** *Best practices for Hybrid Knowledge Bases* **** — Proven patterns for balancing data freshness, coverage, and operational costs
 2. **** *Manage Knowledge Bases in Gen AI Builder* **** — Operational workflows: re-embedding strategies, source lifecycle management, and access controls
 3. **** *Create a Retriever in Gen AI Builder* **** — Hands-on: Fine-tune search behavior, implement advanced filtering, and optimize ranking algorithms
-

3. Model Serving with KServe

- Deploy and scale GPU-powered AI models*

What You'll Learn:

- Deploy high-performance models on GPU infrastructure
- Optimize resource allocation and runtime configurations
- Manage the complete model deployment lifecycle

Time Investment: 45-60 minutes

Modules:

1. ***Model Serving Concepts*** ** — Understand KServe architecture and deployment patterns
 2. ***How-To Configure a ClusterServingRuntime*** ** — Define optimized runtimes for model families with shared configurations
 3. ***How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe*** ** — Hands-on: Leverage NVIDIA NIM for accelerated inference performance
 4. ***How-To Update GPU Resources for an InferenceService*** ** — Hands-on: Right-size GPU and CPU resources for optimal cost-performance
 5. ***How-To Verify InferenceServices and GPU usage*** ** — Validate endpoints and monitor GPU utilization through Hybrid Manager
-

4. Observability & Production Monitoring

- Ensure reliable, monitored AI operations*

What You'll Learn:

- Implement comprehensive observability for AI pipelines
- Monitor model performance, resource usage, and system health
- Establish production readiness and SLA compliance

Time Investment: 20-30 minutes

Modules:

1. ***Observability for Model Serving*** ** — Configure metrics, logging, and alerting for models and data pipelines
 2. ***How-To Monitor deployed models with KServe*** ** — Track endpoint health, latency, and throughput in production
 3. ***Observability for Model Serving*** ** — Leverage platform dashboards and automated alerting systems
-

Continue Your Learning Journey

What's Next?

Ready for Advanced Patterns? Progress to ** *Advanced AI Factory Usage and Extensibility - AI Factory 301* ** to master:

- Multi-agent orchestration and complex workflows
- Advanced embedding pipelines and vector optimization
- Enterprise governance and compliance frameworks
- Scaling patterns for high-volume AI applications

Additional Resources

Deepen Your Understanding:

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
 - Hybrid Manager: Gen AI Builder Guide
 - *Sovereign AI Implementation Guide*
 - *Model Serving Concepts*
 - *Structures Reference Manual*
-

Ready to Build Enterprise AI?

Complete this 201 path to gain the expertise needed for deploying **production-grade Sovereign AI applications** with full control over your models, data pipelines, and operational infrastructure using **EDB Postgres AI**.

Start your first module: *Assistants Reference Manual*

Building Production-Ready AI Features - AI Factory 201

Who this is for

- Developers integrating Gen AI features into production applications
 - Data teams building pipelines for AI-driven Knowledge Bases
 - AI and platform teams configuring model serving and observability
 - Hybrid Manager users enabling Sovereign AI workloads at scale
-

General Goals

By completing this path, you will:

- Build **multi-step AI Assistants** and advanced Structures
- Design **Hybrid Knowledge Bases** and multi-source RAG pipelines
- Deploy and manage **GPU-powered Model Serving**
- Implement **observability and monitoring** for AI-driven features
- Learn patterns for production-grade governance and performance tuning

Modules by Focus Area

1. Advanced Assistant & Structure Design

Goals:

- Implement Assistants with complex personas and memory
- Create advanced multi-step Structures
- Integrate Tools and external data flows

Estimated Time: ~30–45 min

Modules:

1. *Assistants Reference Manual* — Design assistants with personas, memory, and tool usage patterns.
 2. *Structures Reference Manual* — Learn how to chain retrieval and actions into reliable flows.
 3. *Create an Assistant in Gen AI Builder* — Build a production-ready assistant with tools and KBs.
 4. *Create a Structure in Gen AI Builder* — Implement multi-step workflows with validations and branching.
 5. *Create a Tool in Gen AI Builder* — Wrap internal services and APIs as safe, governed tools.
-

2. Data Engineering & Hybrid Knowledge Bases

Goals:

- Design and manage Hybrid Knowledge Bases
- Tune multi-source RAG pipelines
- Implement metadata filtering and hybrid search

Estimated Time: ~30–45 min

Modules:

1. *Best practices for Hybrid Knowledge Bases* — Patterns to balance freshness, coverage, and cost.
 2. *Manage Knowledge Bases in Gen AI Builder* — Operate KBs: re-embedding, source lifecycle, and access control.
 3. *Create a Retriever in Gen AI Builder* — Tune search behavior, filters, and ranking at scale.
-

3. Model Serving with KServe

Goals:

- Deploy GPU-powered models
- Tune runtime and resource settings
- Understand the Model Serving lifecycle

Estimated Time: ~45–60 min

Modules:

1. *Model Serving Concepts*
2. *How-To Configure a ClusterServingRuntime* — Define runtimes to host families of models with shared settings.
3. *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe* — Use NVIDIA NIM images for high-performance inference.
4. *How-To Update GPU Resources for an InferenceService* — Right-size GPU/CPU for performance and cost.
5. *How-To Verify InferenceServices and GPU usage* — Validate endpoints and GPU usage in Hybrid Manager.

4. Observability & Monitoring

Goals:

- Implement observability for AI pipelines and Model Serving
- Monitor performance and resource usage
- Enable production readiness checks

Estimated Time: ~20–30 min

Modules:

1. *Observability for Model Serving* — Metrics and logs for models and pipelines.
2. *How-To Monitor deployed models with KServe* — Track health and performance of serving endpoints.
3. *Observability for Model Serving* — Platform-level dashboards and alerts.

Next steps

After completing this 201 Path:

- Continue to *Advanced AI Factory Usage and Extensibility - AI Factory 301* — advanced patterns for scaling AI apps, multi-agent orchestration, embedding pipelines, and advanced governance.

Related learning resources

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- Hybrid Manager: Using Gen AI Builder
- *Sovereign AI Explained*
- *Model Serving Concepts*
- *Structures Reference Manual*

By mastering this 201 Path, you'll be ready to deploy and scale **Sovereign AI** applications — with full control over your models, pipelines, and production observability — using **EDB PG AI**.

Advanced AI Factory Usage and Extensibility - AI Factory 301

Who this is for

- Advanced developers building **AI-first applications** and agents
- AI OPs teams managing **large-scale model serving**
- Platform engineers building **extensions and integrations** with AI Factory
- Hybrid Manager users building **enterprise-grade Sovereign AI** solutions

General Goals

By completing this path, you will:

- Build complex **agentic Assistants** with Tool calling and advanced RAG pipelines
- Extend **Model Serving** with custom runtimes, Transformers, and Explainers
- Automate and orchestrate **AI Factory pipelines** with APIs
- Design **advanced observability and governance** for production AI
- Integrate AI Factory with **enterprise systems** and existing AI pipelines

Modules by Focus Area

1. Agentic Assistants and Advanced Tools

Goals:

- Build Assistants that reason and act with multi-step Tool usage
- Implement advanced Tools for external API integration
- Chain Tools and manage context flow

Estimated Time: ~45–60 min

Modules:

1. *Assistants Reference Manual* — Architect assistants that reason, retrieve, and act across tools.
2. *Tools Reference Manual* — Design governed actions that call internal or external systems.
3. *Create a Tool in Gen AI Builder* — Implement robust, reusable tool integrations for agent workflows.
4. *Structures Reference Manual* — Build multi-step pipelines and agent behaviors.

2. Extending Model Serving

Goals:

- Create custom **ServingRuntime** definitions for specialized models
- Add **Transformers** and **Explainers** to Model Serving pipelines

- Understand advanced deployment patterns for KServe-based serving

Estimated Time: ~45–60 min

Modules:

1. *Model Serving Concepts*
2. *How-To Configure a ClusterServingRuntime* — Create custom runtimes for specialized models and settings.
3. *How-To Deploy AI Models from Model Library (placeholder — expand with custom Transformers/Explainers patterns)*

3. Observability and Governance at Scale

Goals:

- Implement enterprise-grade **observability** for AI pipelines and Model Serving
- Monitor **Tool usage** and pipeline performance
- Build custom dashboards for hybrid AI workloads

Estimated Time: ~30–45 min

Modules:

1. *Observability for Model Serving* — Metrics, logs, and traces for serving and pipelines.
2. *How-To Monitor deployed models with KServe* — Health and performance of serving endpoints.
3. *Observability for Model Serving*

4. Automation and API Integration

Goals:

- Use **Hybrid Manager and AI Factory APIs** to automate workflows
- Integrate AI Factory pipelines into CI/CD and AI systems
- Manage large-scale AI Factory deployments via API-driven control

Estimated Time: ~20–30 min

Modules:

1. *Hybrid Manager API Overview*
2. *Using the Hybrid Manager API*

Next steps

After completing this 301 Path:

- You will be equipped to design **Sovereign AI applications** at enterprise scale.
- You will understand advanced integration patterns with AI Factory, Hybrid Manager, and existing AI pipelines.
- You will be ready to contribute advanced workflows and reusable components (Tools, Structures, Transformers) to your AI Factory ecosystem.

Related learning resources

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Sovereign AI Explained*
- Hybrid Manager: Using Gen AI Builder
- *Structures Reference Manual*
- *Model Serving Concepts*

By mastering this 301 Path, you will be ready to drive **advanced AI innovation** across your organization — fully under your control — using **EDB PG AI** and **Hybrid Manager**.

AI Factory Explained

Welcome to **AI Factory Explained** — your deep dive into the architecture, concepts, and components of **EDB Postgres® AI** (EDB PG AI).

EDB PG AI enables **Sovereign AI** — building intelligent applications and data products on top of your data, in your databases and data centers, with Postgres as the foundation. The platform integrates vector search, Pipelines, Gen AI Builder, Knowledge Bases, and Model Serving to give you complete control over how AI is embedded in your systems.

This section explains how it all works:

- What key concepts underlie EDB PG AI
- How the architecture fits together
- How individual capabilities like Pipelines, Gen AI Builder, and Model Serving work
- Where vector search and AI models fit within your Postgres estate

Core Concepts

Start here to get oriented to the platform:

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *AI Factory terminology*
- *Fundamental AI Concepts*

Gen AI and Knowledge Bases

Understand how intelligent knowledge is built on your data:

- *Knowledge Bases Reference Manual*
- *Retrievers Reference Manual*
- *Rulesets Manual*
- *Structures Reference Manual*
- *Threads Reference Manual*
- *Tools Reference Manual*

Data Pipelines and AI Accelerator

Understand how Pipelines transform your AI data workflows:

- [AI Accelerator Pipelines overview](#)
- [Data Lake Reference Manual](#)

Vector Search and PGvector

How vector search is embedded directly in Postgres:

- PGvector and Vector Engine concepts

Model Serving

How AI models are deployed, governed, and used:

- [Model Serving Concepts](#)
- [Model Library Explained In Depth](#)

Where to go next:

If you're ready to build:

- Start with a [Learning Paths](#)
- Follow a [Gen AI How-To Guides](#)

Or explore more advanced **solution patterns** in [Industry Solutions with AI Factory](#) and real-world [AI Factory Use Cases](#).

This section grows as EDB PG AI evolves — check back for the latest explanations as new capabilities are added.

AI Factory How-To Guides

Use these guides to deploy and operate AI Factory capabilities. When you run in Hybrid Manager (recommended), follow the HM-specific notes in each guide.

Topics

Gen AI

- Configure Data Sources in Gen AI Builder — see Gen AI references in the hub
- Configure the Data Lake in Gen AI Builder — see Gen AI references in the hub

Model Serving

- [How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe](#)
- [How-To Monitor deployed models with KServe](#)
- [How-To Update GPU Resources for an InferenceService](#)
- [How-To Configure a ClusterServingRuntime](#)

- *Accessing KServe endpoints (internal and external)*

Model Library

- *How-To Integrate Private Registry with Model Library*
- *How-To Define Repository Rules for Model Library*
- *How-To Manage Repository and Image Tag Metadata*
- *How-To Deploy AI Models from Model Library*

Learn more

For conceptual background, see:

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Model Serving Reference Manual*

Learning Paths

AI Factory Learning Paths provide structured guidance to help you build **AI-powered features** and **intelligent applications** using AI Factory — with full control and governance.

Whether you're just starting or scaling **Sovereign AI** solutions in production, these paths guide you through key concepts, tools, and practical implementation — building on your experience as you progress.

How to use the Learning Paths

Each path is **self-paced** and modular.

You can:

- Complete paths **sequentially** (101 → 201 → 301), or
- Dive into topics based on your **role** and **project needs**.

For each path:

- Review the **prerequisites** to ensure readiness.
- Follow the learning flow of:
- **Concepts → How-To Guides → Tutorials → Practice**
- Use your existing AI Factory environment or a sandbox project to practice hands-on.

Learning Paths

Getting Started - AI Factory 101

Audience: New users, developers, data engineers, architects. **Estimated time:** 1–2 hours. **Prerequisites:** Familiarity with basic AI concepts and using web applications.

You will learn to:

- Understand core AI Factory concepts
- Navigate AI Factory and Hybrid Manager
- Create your first AI Assistants
- Connect Knowledge Bases and Retrievers
- Run and review interaction Threads
- Understand key AI Factory terminology

Getting Started - AI Factory 101

Building Production-Ready AI Features - AI Factory 201

Audience: Developers, data engineers, solution architects building production AI features. **Estimated time:** 2–4 hours. **Prerequisites:** Completion of 101 Path, basic knowledge of Kubernetes concepts.

You will learn to:

- Architect **hybrid Knowledge Bases** for advanced search
- Use and manage **Rulesets** and governance patterns
- Build **advanced Assistants** and multi-source Retrievers
- Implement **GPU-powered Model Serving** with KServe
- Apply monitoring and **observability best practices**
- Implement production-readiness and scaling strategies

Building Production-Ready AI Features - AI Factory 201

Advanced AI Factory Usage and Extensibility - AI Factory 301

Audience: AI platform owners, advanced developers, AI engineers, architects. **Estimated time:** 4–6 hours (self-paced, advanced topics). **Prerequisites:** Completion of 201 Path, experience with Kubernetes and container-based AI workloads.

You will learn to:

- Design **Agentic Assistants** and advanced Structures
- Develop and deploy **custom Tools**
- Extend Model Serving with **custom ServingRuntimes**
- Implement **model explainability** and responsible AI patterns
- Automate AI Factory pipelines via **API-driven workflows**
- Apply **advanced observability** and performance tuning

Advanced AI Factory Usage and Extensibility - AI Factory 301

Recommended Training Courses

To complement these self-paced Learning Paths, we also offer:

Instructor-Led Training

- Advanced AI Factory Architectures
- AI Factory Administration & Operations
- Custom Model Development & Deployment with AI Factory

Self-Paced Training

- Introduction to AI Factory
- Building AI Assistants with AI Factory
- Managing AI Models and Hybrid Knowledge Bases
- Scaling AI Workloads with AI Factory

Where to next?

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Gen AI How-To Guides*
- *Model Serving Quickstart*

Industry Solutions with AI Factory

AI Factory provides a flexible platform for building **trusted, scalable AI-powered applications** across industries — combining your enterprise data, AI pipelines, model serving, and conversational agents.

This page highlights **common solution patterns** and **examples by industry** — along with links to relevant *Gen AI How-To Guides* and *Learning Paths* to help you build them.

For more technical patterns, see *AI Factory Use Cases* .

Financial Services

AI Factory helps financial services organizations deliver **secure, compliant AI** for customer experience and internal operations.

Example goals:

- Improve customer service with **automated agents** grounded in proprietary knowledge.
- Enable secure, explainable **internal knowledge search** for advisors and staff.
- Automate document-heavy workflows (KYC, onboarding).

Example solutions:

- **Client Support Chatbots** powered by *Create an Assistant in Gen AI Builder* , backed by *Create a Knowledge Base in Gen AI Builder* and governed by *Create a Ruleset in Gen AI Builder* .
 - **Internal Knowledge Search** combining *Create a Retriever in Gen AI Builder* and *Create a Knowledge Base in Gen AI Builder* , optionally enriched via Pipelines and data preparation .
 - **Automated Document Processing** with Pipelines and data preparation , using *Preparers perform OCR operation examples* , *Preparers summarize text operation examples* , and AI-driven extraction.
-

Healthcare and Life Sciences

AI Factory supports **regulated, compliant AI** for clinical and operational use cases.

Example goals:

- Improve clinical decision support with **literature-aware agents**.
- Automate **summarization** of complex medical documents.
- Deliver personalized **patient-facing assistants**.

Example solutions:

- **Medical Literature Search** using *Create a Knowledge Base in Gen AI Builder* , optimized *Create a Retriever in Gen AI Builder* , and conversational *Create an Assistant in Gen AI Builder* .
 - **Clinical Document Summarization** with document prep via Pipelines and data preparation and high-performance inference using *Model Serving Reference Manual* .
 - **Patient Support Assistants** built with *Create an Assistant in Gen AI Builder* , compliant *Create a Ruleset in Gen AI Builder* , and external integrations via *Create a Tool in Gen AI Builder* .
-

Retail and E-Commerce

AI Factory enables **personalized, AI-powered experiences** for retail and e-commerce applications.

Example goals:

- Deliver **semantic product search** experiences.
- Power **personalized recommendations** across channels.
- Automate **customer support workflows**.

Example solutions:

- **Semantic Product Search** with embeddings served via *Model Serving Reference Manual* , queried using tuned *Create a Retriever in Gen AI Builder* .
- **Personalized Recommendations** combining *Create an Assistant in Gen AI Builder* , *Create a Retriever in Gen AI Builder* , and **real-time API integrations** with *Create a Tool in Gen AI Builder* .
- **Customer Service Automation** with conversational *Create an Assistant in Gen AI Builder* , *Create a Knowledge Base in Gen AI Builder* , and brand-safe *Create a Ruleset in Gen AI Builder* .

Manufacturing and Industrial

AI Factory brings **AI-driven knowledge and automation** to the shop floor and field operations.

Example goals:

- Build **maintenance assistants** that surface engineering knowledge.
- Enable **search across technical manuals and SOPs**.
- Automate **document understanding** and reporting.

Example solutions:

- **Knowledge-driven Maintenance Assistants** built with *Create an Assistant in Gen AI Builder* , connected to *Create a Knowledge Base in Gen AI Builder* , and powered by API-based *Create a Tool in Gen AI Builder* .
 - **Document Search across Manuals and SOPs** using Pipelines and data preparation for document processing, and RAG with tuned *Create a Retriever in Gen AI Builder* .
-

Public Sector and Government

AI Factory helps governments and public sector organizations deliver **transparent, auditable AI** in sensitive contexts.

Example goals:

- Support **internal policy search and compliance checks**.
- Power **citizen-facing service portals**.
- Enable **knowledge-grounded AI experiences** with governance.

Example solutions:

- **Internal Policy Search and Compliance QA** with trusted *Create a Knowledge Base in Gen AI Builder* , *Create a Retriever in Gen AI Builder* , and auditable *Create an Assistant in Gen AI Builder* .
 - **Citizen Support Portals** built with conversational *Create an Assistant in Gen AI Builder* , external system integrations via *Create a Tool in Gen AI Builder* , and governed *Create a Ruleset in Gen AI Builder* .
-

Build Your Solution

AI Factory gives you a **modular architecture** to combine:

- **** *Create an Assistant in Gen AI Builder* **** and **Agent Studio** for conversational AI.
- **** *Create a Knowledge Base in Gen AI Builder* **** and **** *Create a Retriever in Gen AI Builder* **** for RAG and semantic search.
- **** *Create a Ruleset in Gen AI Builder* **** for governance and compliance.
- **** *Create a Tool in Gen AI Builder* **** for connecting external systems and business logic.

- **** *Model Serving Reference Manual* **** for scalable, production-ready inference.
 - **** Pipelines and data preparation **** for preparing data for AI-driven applications.
-

Learn More

- *Gen AI How-To Guides* — Step-by-step building blocks
- *Learning Paths* — Structured skill-building journeys
- *AI Factory Use Cases* — Common technical patterns to adapt to your solution

→ Ready to get started? Pick a *Learning Paths* or explore relevant *Gen AI How-To Guides* .

AI Factory gives you the foundation to bring **trusted, scalable AI** to your industry — securely and under your governance.

AI Factory Use Cases

AI Factory is designed to help you build **Sovereign AI** solutions — where your **data, models**, and **AI-driven applications** operate under your governance, in your infrastructure, with your trusted content.

These use cases highlight how AI Factory components work together to support **real-world, production-ready Sovereign AI applications** — across industries and solution patterns.

For industry-specific solution examples, see: *Industry Solutions with AI Factory* .

Retrieval-Augmented Generation (RAG)

RAG is a foundational pattern for Sovereign AI — it lets you deliver **accurate, grounded responses** by combining **LLMs** with your enterprise data.

What

- Use AI models that **retrieve trusted content** from your Knowledge Bases, rather than relying only on opaque base models.
- Make responses **auditable** and **explainable** — key for Sovereign AI.

Why

- Reduce hallucinations.
- Maintain compliance and traceability.
- Ensure AI answers reflect your latest business knowledge.

How

- Build *Create a Knowledge Base in Gen AI Builder* from your Data Sources.
- Implement tuned *Create a Retriever in Gen AI Builder* for semantic search.
- Power *Create an Assistant in Gen AI Builder* and apps with RAG pipelines.

Learn More

- *Sovereign AI Explained*
 - *Best practices for Hybrid Knowledge Bases*
-

Document Intelligence Pipelines

Automate the transformation of documents into **structured, queryable knowledge** that remains **inside your governance boundary**.

What

- Apply **OCR, parsing, summarization, and metadata extraction**.
- Store results in *Create a Knowledge Base in Gen AI Builder* or **Vector Engine**.

Why

- Keep sensitive documents in your infrastructure — no third-party API calls required.
- Build explainable pipelines for compliance-heavy domains.

How

- Use Pipelines and data preparation .
- Leverage built-in *Pipelines reference - Preparers* (OCR, parse PDF, parse HTML).
- Index results for search or RAG.

Learn More

- *Sovereign AI Concepts*
 - Pipeline Getting Started
 - *Document Processing Examples*
-

Conversational Assistants and Chatbots

Build Assistants that run **within your infrastructure**, use **your models**, and rely on **your Knowledge Bases** — with complete visibility and control.

What

- AI Assistants powered by **Model Serving** and your **RAG pipeline**.
- Defined and governed with *Create a Ruleset in Gen AI Builder* .

Why

- Avoid relying on public API services for LLM responses.
- Control tone, style, compliance — essential for Sovereign AI.

How

- Build *Create an Assistant in Gen AI Builder* in Agent Studio.
- Govern behavior with *Rulesets (optional)* .
- Add *Create a Tool in Gen AI Builder* to safely integrate internal systems.

Learn More

- Agent Studio in Hybrid Manager
- *Sovereign AI Explained*
- *Getting Started - AI Factory 101*

Semantic Search Across Enterprise Content

Enable **natural language search** on your content — while ensuring embeddings and search data remain **inside your control**.

What

- Vector-based search on **Knowledge Bases** or **Vector Engine**.
- Supports text, hybrid structured/unstructured data, images.

Why

- Keep embeddings in your databases or object storage — not in third-party vector DBs.
- Maintain auditability and security.

How

- Build *Create a Knowledge Base in Gen AI Builder* .
- Use *Create a Retriever in Gen AI Builder* to expose search to Assistants or apps.
- Use *Vector Engine (PGvector)* for in-database search.

Learn More

- Vector Engine Hub
- *Knowledge Bases Reference Manual*
- *Building Production-Ready AI Features - AI Factory 201*

Real-Time Model Inference APIs

Deploy and serve models **inside your Kubernetes clusters**, under your governance — not through opaque third-party APIs.

What

- Serve LLMs, embeddings, vision models via **KServe-based Model Serving**.
- Models pulled from your *Model Library Reference Manual* , deployed via *How-To Create an InferenceService for NVIDIA NIM Container* .

Why

- Full control of model stack and model versions.
- Run inference in **GPU-accelerated, air-gapped environments** if needed.

How

- Manage models in *Model Library Reference Manual* .
- Deploy with *Model Serving Reference Manual* .
- Secure endpoints with Hybrid Manager controls.

Learn More

- *Model Serving Concepts*
- *Sovereign AI Explained*
- *Advanced AI Factory Usage and Extensibility - AI Factory 301*

Automated AI-Powered ETL Pipelines

Keep **AI data pipelines** inside your controlled environment — with no data leakage.

What

- Use *Create a Structure in Gen AI Builder* and Pipelines to run:
- AI categorization
- Document summarization
- Metadata extraction
- Data cleansing

Why

- Avoid sending sensitive data to cloud APIs.
- Build reusable, explainable ETL components.

How

- Build *Create a Structure in Gen AI Builder* .
- Run as Tools in *Create an Assistant in Gen AI Builder* or standalone Pipelines.
- Monitor and govern pipeline execution via Hybrid Manager.

Learn More

- *Structures Reference Manual*
- *Advanced AI Factory Usage and Extensibility - AI Factory 301*

Summary

Sovereign AI use cases are about keeping **data, models, and AI applications under your governance**:

- Build **RAG pipelines** grounded in your enterprise data
- Automate document intelligence workflows
- Deploy conversational Assistants in your infrastructure
- Serve AI models via GPU-backed Model Serving
- Run AI-powered ETL pipelines with no data leaving your environment
- Power enterprise semantic search with full control of embeddings

Next steps:

- Explore *Industry Solutions with AI Factory*
- Follow a *Learning Paths*
- Get started with *Gen AI How-To Guides*

AI Factory gives you the foundation to build **trusted, governed, explainable Sovereign AI** — on your terms.

→ Start today with *Getting Started - AI Factory 101* , or deploy your first *Create an Assistant in Gen AI Builder* .

2.3 Concepts

EDB Postgres® AI (EDB PG AI) brings **AI workloads into your Postgres platform and infrastructure** — with an architecture designed to power **Sovereign AI** on hybrid and cloud-native environments.

AI Factory extends Postgres to support:

- Vector search and semantic search
- Data pipelines for embedding generation
- Gen AI application development
- Scalable model serving and inference
- AI-driven governance and observability

AI Factory helps you build and operate **trusted, enterprise-grade AI systems** using your data, models, and applications.

2.4 Concepts

2.4.1 EDB's vision for AI with Postgres and Hybrid Manager

EDB's AI Factory strategy makes Postgres a **first-class platform** for AI workloads and AI-driven applications.

The architecture combines:

- **AI in Postgres:** vector search, embedding pipelines, and AI functions within the database
- **Gen AI apps:** conversational and task-driven AI applications with Gen AI Builder
- **Scalable inference:** GPU-accelerated model serving using Kubernetes and KServe
- **Unified control plane:** model lifecycle management and observability in Hybrid Manager

This enables **Sovereign AI** — AI on your data, on your infrastructure, with full governance.

2.5 Concepts

RAG pipelines enhance LLM responses by combining:

- Domain-specific embeddings from *Knowledge Bases Reference Manual*
- Configurable search behavior with *Retrievers Reference Manual*
- Scalable model inference through *Model Serving Reference Manual*
- Application orchestration via *Gen AI Builder*

This enables more accurate, grounded, and compliant AI applications.

2.6 Concepts

AI Factory provides robust model serving with:

- Scalable inference services for LLMs, embedding models, and vision models
- GPU-accelerated serving via Kubernetes + KServe
- Flexible deployment options through the *Model Serving Reference Manual* framework
- Lifecycle management and discoverability with the *Model Library Explained In Depth*

These services power both **Gen AI Builder agents** and **Knowledge Bases**.

2.7 Concepts

AI Factory Pipelines and data preparation automate:

- Embedding generation
- Chunking, parsing, summarizing content
- Keeping embeddings fresh with *Pipeline auto-processing*

Explore *Pipelines reference - Preparers* for reusable data preparation components.

2.8 Concepts

2.8.1 Summary

EDB PG AI enables you to:

- Embed AI capabilities directly into Postgres
- Build scalable, production-ready AI applications
- Operate Sovereign AI on your infrastructure with unified governance
- Integrate open-source tools and standards across the AI stack

Next steps:

- [AI Factory terminology](#)
 - [Fundamental AI Concepts](#)
 - [AI Factory Learning Center](#)
-

AI Factory vs. “Generic AI”

AI Factory focuses on Gen AI and data infrastructure that run with your Postgres estate and, primarily, within EDB Hybrid Manager. “Generic AI” refers to general-purpose AI/ML stacks and services you might already use — for example, external LLM APIs, ML platforms, and data science tooling.

This page explains who each approach is for, what each does best, where they run, when to choose one or the other, and how they can work together. It also answers common questions we hear from teams adopting EDB Postgres® AI.

What we mean by each

- AI Factory: A modular set of capabilities for Gen AI and intelligent data apps on your Postgres data. It includes [Gen AI Builder](#) , Pipelines and data preparation , [Vector Engine \(PGvector\)](#) , and [Model Serving Reference Manual](#) . It is designed for Hybrid Manager first, where you gain a unified control plane for governance and observability via [AI Factory in Hybrid Manager](#) .
- Generic AI: General-purpose AI and ML tooling outside of AI Factory — such as cloud LLM APIs or standalone ML platforms used for experimentation and model development. You can combine these with AI Factory to build end-to-end solutions.

For an overview of concepts and architecture, see [Gen AI Builder Concepts \(Assistants, Knowledge, Tools, Threads\)](#) and [AI Factory Explained](#) .

Who it is for

- AI Factory: Application developers, data engineers, and platform teams who want to build Gen AI assistants, document intelligence, and AI-powered features on trusted Postgres data with governance. See [Learning Paths](#) to get hands-on.
 - Generic AI: Data science and research teams doing model exploration, fine-tuning, or domain R&D on specialized ML tooling. These efforts can feed models and components into AI Factory for operationalization.
-

Where it runs

- Hybrid Manager (hub): Most AI Factory capabilities are delivered and managed within Hybrid Manager's AI Factory . This gives you lifecycle control, governance, and observability across Gen AI, pipelines, vector search, and model serving.
 - Outside Hybrid Manager (selected components): You can run Postgres-focused capabilities such as *Vector Engine (PGvector)* , Pipelines and data preparation , or AIDB features where supported. When you do so, you own cluster lifecycle, security, and observability. For details, see *Where AI Factory runs* .
-

When to choose each

Choose AI Factory when you:

- Build assistants, search, or workflow automation that rely on enterprise data in Postgres.
- Need governed Retrieval-Augmented Generation (RAG) with Pipelines and data preparation and *Vector Engine (PGvector)* .
- Want unified observability and lifecycle management (Hybrid Manager).
- Plan to serve models behind stable endpoints via *Model Serving Reference Manual* .

Choose Generic AI when you:

- Experiment with model architectures, training data, or evaluation workflows in a research setting.
- Use external APIs for prototyping or transient workloads that do not require integration with Postgres governance.
- Run batch model training that is not part of your operational Postgres or Hybrid Manager environment.

Many organizations do both: Generic AI for exploration, AI Factory for productionizing data-centric AI on Postgres.

How they work together

- Bring your own models: Package a vetted model and deploy it as an inference service with *Model Serving Reference Manual* . Use the *Model Library Reference Manual* patterns to organize model assets.
 - Use external providers: Build assistants and tools in *Gen AI Builder* that can call external services through drivers. See Drivers overview .
 - Operationalize data flows: Use Pipelines and data preparation to prepare, transform, and keep content fresh for vector search and RAG.
 - Query with vectors in Postgres: Store and search embeddings with *Vector Engine (PGvector)* alongside your relational data for simpler operations and security.
-

Use cases

- Enterprise knowledge assistants: Build domain assistants that retrieve trusted answers from Knowledge Bases and Postgres. Start with *Gen AI Builder* and *Vector Engine (PGvector)* .
- Document intelligence: Classify, extract, and summarize documents using Pipelines and data preparation integrated with Gen AI assistants.

- AI-powered APIs for apps: Serve embeddings, recommendations, or language tasks behind *Model Serving Reference Manual* .
 - Model exploration and evaluation: Use your preferred “Generic AI” tooling to evaluate candidate models before promoting them into AI Factory for serving.
-

Prerequisites

- Using Hybrid Manager: Access to a Hybrid Manager instance with AI Factory enabled and entitlements for Gen AI, pipelines, vector search, and model serving. See *AI Factory in Hybrid Manager* .
- Running components outside Hybrid Manager: A supported Postgres environment for extensions and AIDB, and the infrastructure required for pipelines and serving where applicable. Validate platform compatibility, security, and licensing.

If you need precise support matrices for outside-HM usage, coordinate with your EDB representative.

FAQ

Q: Does AI Factory train models?

A: AI Factory focuses on Gen AI and data infrastructure — retrieving, augmenting, and serving models with your data. Use your Generic stack for model training and experimentation, then serve vetted models via *Model Serving Reference Manual* .

Q: Can I run AI Factory without Hybrid Manager?

A: Many capabilities are intended to run in Hybrid Manager. Selected components — such as *Vector Engine (PGvector)* , Pipelines and data preparation , and AIDB features — can run outside Hybrid Manager when supported. See *Where AI Factory runs* .

Q: How do I bring my own LLM or embedding model?

A: Package and deploy it using *Model Serving Reference Manual* . Manage artifacts and metadata using the library patterns under *Model Library Reference Manual* .

Q: How do assistants access enterprise data safely?

A: Use *Gen AI Builder* with governed Knowledge Bases, RAG via Pipelines and data preparation , and Postgres-native *Vector Engine (PGvector)* . In Hybrid Manager, these workloads run under unified governance and observability.

Q: Where do I start?

A: Review *AI Factory Concepts* and pick a *Learning Paths* . If you operate in Hybrid Manager, start from AI Factory in Hybrid Manager .

Related topics

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *AI Factory Explained*
- *Gen AI Builder*
- Pipelines and data preparation
- *Vector Engine (PGvector)*

- *Model Serving Reference Manual*
- *AI Factory in Hybrid Manager*

Fundamental AI Concepts

AI Factory builds on modern AI concepts, architectures, and technologies to enable advanced data-driven and intelligent applications.

This page explains key **industry concepts** that form the foundation of AI Factory's design and capabilities:

- AI concepts
- Vector and semantic search
- Retrieval-augmented generation (RAG)
- AI for databases
- AI for infrastructure
- Hybrid and Sovereign AI architectures

To see how these concepts are implemented in AI Factory and Hybrid Manager, start with:

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *AI Factory terminology*

Before you start

You'll get the most out of this section if you have:

- A basic understanding of **AI workflows**
- Familiarity with **vector databases** and **semantic search**
- Awareness of **AI pipelines** and **LLM architectures**
- Helpful: experience with **AI Factory Concepts** and *Getting Started - AI Factory 101*

Core AI concepts

Learning systems and model training

Learning systems enable models to learn from data and improve performance on specific tasks without explicit programming.

Common learning types:

- **Supervised learning** — Predict based on labeled data (ex: sales forecasting)
- **Unsupervised learning** — Find patterns in unlabeled data (ex: customer segmentation)
- **Reinforcement learning** — Learn through interaction and reward feedback (ex: game-playing agents)

Infrastructure needs:

- Scalable storage for training data
- Compute for model training (CPUs, GPUs, TPUs)
- Optimized serving infrastructure for inference

Deep learning (DL) & neural networks

DL uses deep neural networks with multiple layers to extract and learn complex patterns.

Strengths:

- Handles unstructured data (images, text, audio, video)
- Powers advanced applications such as image recognition, speech recognition, text generation

Infrastructure needs:

- High-performance compute (GPUs or TPUs)
 - Optimized data pipelines and model serving infrastructure
-

Natural language processing (NLP)

NLP applies AI to understand and generate human language.

Common use cases:

- Text classification
- Sentiment analysis
- Machine translation
- Conversational AI
- Intelligent document processing
- Content summarization

Infrastructure needs:

- Large model architectures (Transformers)
 - Low-latency serving infrastructure for real-time interaction
 - Data pipelines to feed and fine-tune models
-

Large language models (LLMs)

LLMs are advanced NLP models based on deep learning architectures such as Transformers.

Key characteristics:

- Trained on massive text corpora
- Generate coherent, context-aware text
- Support chatbots, content generation, code assistants, advanced Q&A

LLMs are the foundation for modern Gen AI applications.

Infrastructure needs:

- Enormous compute and storage for training
- GPU-optimized serving for real-time inference

- Scalable infrastructure for multi-user workloads

Embeddings and vector representations

Embeddings are dense vector representations of data, capturing semantic meaning in numerical form.

Applications:

- Semantic search
- Personalization and recommendations
- Anomaly detection
- RAG (retrieval-augmented generation)

How embeddings are used:

- Convert text, images, audio, video, or structured data into vectors
- Store in vector databases or Postgres with pgvector
- Perform similarity search using approximate nearest neighbor (ANN) algorithms

Vector databases and semantic search

Databases optimized to store and search embeddings.

Key features:

- Fast vector similarity search
- High-dimensional vector storage
- Scale to billions of vectors
- Support for hybrid search (vector + keyword / filter)

Common use cases:

- Semantic search engines
- Personalized recommendations
- Knowledge assistants with grounding
- Fraud detection and anomaly detection
- Enhanced Gen AI pipelines (RAG)

Retrieval-augmented generation (RAG)

RAG combines vector search with LLMs to generate grounded responses.

How it works:

1. User query → converted to embedding
2. Vector search retrieves relevant documents / content
3. Retrieved context is injected into LLM prompt

4. LLM generates a response using both its model knowledge and retrieved context

Benefits:

- More accurate and grounded AI responses
- Real-time, dynamic knowledge integration
- Support for domain-specific and compliance-aware AI applications

RAG is a core architecture pattern supported by AI Factory.

AI for databases

Intelligent database management

Using AI to optimize database operations:

- Automated performance tuning
- Intelligent query optimization
- Proactive resource management
- Predictive maintenance
- Self-healing capabilities

Goal: **autonomous, highly optimized database infrastructure**.

In-database AI

Perform model training and inference **inside the database**, reducing data movement.

Benefits:

- Faster insights and response times
- Simplified architecture
- Lower latency for in-database applications
- Real-time prediction capabilities
- Streamlined model operations workflows

Example: invoking models from SQL in Postgres.

Vector databases in Postgres

AI Factory leverages **pgvector** extension in Postgres:

- Native vector type and operations
- Integration with AI pipelines and embedding models
- Full integration with Hybrid Manager and Model Serving
- Enable semantic search and RAG pipelines on **your Postgres data**

Brings AI capabilities closer to core data.

AI for infrastructure

AI-accelerated hardware (GPUs, TPUs)

AI workloads depend heavily on specialized hardware:

- **GPUs** — General-purpose acceleration for training and inference
- **TPUs** — ASICs optimized for deep learning
- **Other accelerators** — FPGAs, inference chips

Benefits:

- Orders-of-magnitude speedup for model training
- High throughput for production inference
- Efficient resource utilization in cloud-native AI clusters

AI Factory integrates optimized GPU serving via **KServe**.

Leveraging cloud compute for AI workloads

Modern AI workloads rely on cloud compute to scale:

- GPU / TPU instances on demand
- Managed Kubernetes services for training and serving (ex: Hybrid Manager)
- Optimized AI stacks (CUDA, PyTorch, TensorFlow, JAX)
- Distributed model training with advanced networking
- Hybrid and multi-cloud architectures

Goals:

- Maximize flexibility and performance
- Minimize cost
- Enable portability across cloud providers and environments

AI Factory supports **hybrid cloud** and **Sovereign AI** patterns natively.

Hybrid and Sovereign AI

Modern enterprises increasingly require **Sovereign AI**:

- Data remains in your infrastructure and control
- AI pipelines leverage your trusted knowledge and content
- Model serving runs in your infrastructure (ex: Hybrid Manager + KServe)
- Governance, observability, and compliance are built in

AI Factory enables:

- AI in Postgres
- Pipelines and Knowledge Bases with **your data**
- Model Serving in your Kubernetes cluster
- Full visibility and control via Hybrid Manager dashboards

Sovereign AI is a first-class design goal for EDB AI Factory.

Related concepts

- *Fundamental AI Concepts*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*

Next steps

To see how these concepts power **EDB AI Factory**, continue with:

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *AI Factory terminology*
- *Getting Started - AI Factory 101*

AI Factory terminology

AI Factory terminology defines key concepts and technologies used across **EDB Postgres® AI** (EDB PG AI), **Hybrid Manager AI Factory**, and related components.

This page complements these conceptual explanations:

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Fundamental AI Concepts*

Understanding these terms helps you build trusted, governed, and scalable AI solutions with EDB PG AI.

Core AI concepts

Machine learning (ML)

Machine learning uses algorithms that improve through data exposure to perform tasks without explicit programming. It powers predictive analytics, automation, and decision making in AI Factory through *Model Serving Reference Manual*, Pipelines, and Assistants.

[Learn more](#)

- NOTE THIS IS NOT AN INTENDED USE CASE SUPPORTED BY HYBRID MANAGER CURRENTLY *

Deep learning (DL)

Deep learning leverages multi-layer neural networks to model complex data patterns. It supports advanced applications such as language understanding and image recognition within AI Factory, enabled through *Model Serving Reference Manual* with GPU acceleration.

[Learn more](#)

Natural language processing (NLP)

Natural language processing (NLP) enables computers to understand and generate human language. It underpins semantic search and conversational AI in AI Factory via *Knowledge Bases Reference Manual* , *Retrievers Reference Manual* , and *Assistants Reference Manual* .

[Learn more](#)

Large language models (LLMs)

LLMs are large deep learning models trained on massive text corpora. In AI Factory, they drive *Assistants Reference Manual* , Retrieval-Augmented Generation (RAG), and various model pipelines, deployed using *Model Serving Reference Manual* .

[Learn more](#)

Embeddings

Embeddings are vector representations of data that capture semantic meaning. AI Factory Pipelines create embeddings used in *Knowledge Bases Reference Manual* and served through the *Vector Engine (PGvector)* to enable semantic search and RAG.

[Learn more](#)

Vector databases

Vector databases store embeddings and enable fast similarity search. AI Factory provides this through the *Vector Engine (PGvector)* , built on the open-source `pgvector` extension, integrated directly with Postgres.

Retrieval-augmented generation (RAG)

RAG combines vector search with LLM generation to ground model responses in relevant documents. In AI Factory, it is implemented through *Knowledge Bases Reference Manual* , *Retrievers Reference Manual* , and *Model Serving Reference Manual* .

[Intro to RAG](#)

AI for databases

Intelligent database management

Intelligent database management applies AI to optimize Postgres performance and operations. AI Factory extends this with intelligent retrieval and search using *Vector Engine (PGvector)* and Pipelines.

Vector search in Postgres

Vector search allows you to query embeddings directly within Postgres. AI Factory uses [pgvector](#) to power this capability through the *Vector Engine (PGvector)* , supporting Knowledge Bases and RAG.

AIDB

AIDB (AI-in-Database) brings vector search, embedding pipelines, and future capabilities to HM-managed Postgres clusters. It is the foundation for AI Factory Pipelines and data preparation and Knowledge Bases. ===== Intelligent database management applies AI to optimize Postgres performance and operations. AI Factory extends this with intelligent retrieval and search using [Vector Engine](#) and Pipelines.

In-database machine learning (In-DB ML)

In-DB ML enables running vector search and ML pipelines inside Postgres, reducing data movement and latency. AI Factory implements this through [Vector Engine](#) and [Pipelines](#) .

Vector search in Postgres

Vector search allows you to query embeddings directly within Postgres. AI Factory uses [pgvector](#) to power this capability through the *Vector Engine (PGvector)* , supporting Knowledge Bases and RAG.

AIDB

AIDB brings vector search, embedding pipelines, and other capabilities to HM-managed Postgres clusters. It is the foundation for AI Factory [Pipelines](#) and Knowledge Bases.

Natural language interfaces to databases

Natural language interfaces enable users to query Postgres using natural language rather than SQL. AI Factory Assistants leverage this pattern through *Gen AI Builder* , combining LLMs with structured and unstructured retrieval.

AI infrastructure

AI-accelerated hardware

AI Factory uses GPU-accelerated Kubernetes clusters to serve deep learning models and high-throughput inference. Model workloads in [Model Serving](#) run on GPU-enabled nodes.

[Learn more](#)

KServe

KServe is the open-source Kubernetes-native framework AI Factory uses to deploy and manage models. It provides InferenceServices, autoscaling, and observability for AI Factory *Model Serving Reference Manual* .

[KServe documentation](#)

Model Serving

Model Serving deploys AI models as production-grade inference services, using KServe under the hood. It supports LLMs, embedding models, vision models, and custom AI workloads.

See: *Model Serving Reference Manual* , *Model Serving Explained*

Gen AI Builder

Gen AI Builder

Gen AI Builder is the application layer of AI Factory for building AI-powered apps and agents. It integrates:

- *Assistants Reference Manual* for conversational AI
- *Knowledge Bases Reference Manual* for RAG and semantic search
- *Retrievers Reference Manual* for advanced search patterns
- *Tools Reference Manual* to perform external actions
- *Rulesets Manual* to control Assistant behavior
- *Threads Reference Manual* to persist conversation history
- *Structures Reference Manual* to define AI workflows and pipelines

It is deployed on *Hybrid Manager AI Factory* , enabling full-stack Sovereign AI.

Hybrid Manager AI Factory

Hybrid Manager

Hybrid Manager provides the Kubernetes-based control plane for AI Factory workloads, managing GPU resources, Pipelines, Model Serving, and Knowledge Bases. It provides end-to-end governance, security, and observability.

See: *Hybrid Manager AI Factory*

Pipelines

Pipelines automate the creation of embeddings and Knowledge Bases from data sources in your control, ensuring transparency and auditability.

See: *AI Accelerator Pipelines overview*

Knowledge Bases

Knowledge Bases index content for semantic search and RAG. They support multi-source, multi-modal search and power AI Factory Assistants.

See: *Knowledge Bases Reference Manual*

Model Serving

Model Serving deploys models using Kubernetes-native KServe and integrates with the *Model Library Reference Manual* . It powers Assistants, Knowledge Bases, and custom AI applications.

See: *Model Serving Reference Manual*

Additional concepts

Image and Model Library

The Image and Model Library in Hybrid Manager manages container images for both Postgres and AI model deployments. The *Model Library Reference Manual* provides an AI-focused view, supporting Model Serving and governed image workflows.

See: *Model Library Reference Manual*

NVIDIA NIM

NVIDIA NIM Microservices are optimized AI model images from NVIDIA for high-performance inference. AI Factory uses NIM containers for LLMs, embeddings, and vision models, served through Model Serving.

See: *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe*

Summary

Understanding these terms will help you build with:

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *AI Factory Explained*
- *Hybrid Manager AI Factory*

Together, these capabilities enable you to deliver **trusted, governed, Sovereign AI** — with your data, in your databases, under your control.

Sovereign AI Explained

Sovereign AI is a core principle of **EDB Postgres® AI** (EDB PG AI): Giving you full control over your **data, models, and AI stack** — with **auditability, security, and trust** built in.

Rather than relying on opaque cloud APIs and external data flows, **Sovereign AI** ensures that your intellectual property, business data, and AI-driven applications stay under your governance — and run on infrastructure you control.

Why it matters

AI is transforming how organizations interact with data and deliver new experiences. But with AI adoption comes real risks:

- Many public AI APIs are **black boxes** — you lose control over how your data is used.
- AI embeddings and vectors often become **orphaned** in third-party systems.
- Sensitive data may leave compliance boundaries — creating **regulatory risk**.
- AI results may be **non-auditable**, raising concerns in regulated industries.
- Application **traceability** and **explainability** are often missing.

Sovereign AI answers these challenges:

- **Own your data and embeddings** — stored in Postgres and your object storage.

- **Own your models and AI stack** — served through Pipelines and Model Serving.
- **Own your Knowledge Bases** — fully governable, versioned, and auditable.
- **Own your AI workloads** — running in Kubernetes environments you control.
- **Own your Assistant behavior** — defined, governed, and observable in Agent Studio.

What is Sovereign AI?

Sovereign AI is an architecture and operating model where:

- **Data stays inside your trust boundaries.**
- **AI pipelines are fully transparent and auditable.**
- **Model serving happens on infrastructure you manage.**
- **Embeddings are stored where you control them** (Postgres, object storage).
- **Assistant behavior is governed and versioned.**
- **APIs exposed to your users are secured and observable.**
- You can **choose whether and how external APIs are involved** — no hidden dependencies.

Sovereign AI is a spectrum — not an all-or-nothing choice:

- Some organizations will want **fully self-managed, zero external API** architectures.
- Others will use hybrid models — **private Knowledge Bases + public LLMs**, carefully governed.
- EDB PG AI supports this full spectrum — and ensures you are never locked into an external stack.

How EDB PG AI enables Sovereign AI

EDB PG AI is designed from the ground up for Sovereign AI:

- **Data-first architecture** — Knowledge starts in your databases and object storage.
- **Vector Engine** — Your embeddings live in Postgres or controlled storage.
- **Pipelines** — You control every step of the data-to-embedding-to-query pipeline.
- **Model Serving** — Models run on your Kubernetes infrastructure, under your governance.
- **Agent Studio** — Rulesets, Tools, Structures, and Assistants are all governed assets.
- **Observability** — Full audit trails and monitoring of model and Assistant activity.

Key components:

- *Vector Engine (PGvector)* — Vector search in your Postgres clusters.
- Pipelines and data preparation — Controlled creation of Knowledge Bases and embeddings.
- *Model Serving Reference Manual* — Kubernetes-native, GPU-accelerated model inference.
- Gen AI in Hybrid Manager — Governed Assistant and Tool development.

You can also selectively integrate **external LLM APIs** — but **with full visibility and choice**.

Why Hybrid Manager?

Today, EDB PG AI's **strongest Sovereign AI model** is delivered with **Hybrid Manager™** (EDB):

See: Hybrid Manager Gen AI Workloads

Hybrid Manager provides the architecture to:

- Deploy Pipelines and Model Serving on **EDB-optimized Kubernetes**.
- Manage Knowledge Bases with **full lifecycle control**.
- Monitor and trace all AI activity — at the database, Pipeline, and Assistant levels.
- Govern **model deployment**, image provenance, and retriever configurations.
- Enable **secure API access** to Assistants and semantic search.
- Implement fine-grained **data residency and trust boundaries**.

Full stack control — from your Postgres database → embeddings → Pipelines → model inference → Assistants and APIs — all observable and auditable.

Summary

- **Sovereign AI** is about trust, transparency, and control.
- EDB PG AI gives you a complete stack to implement Sovereign AI — with no lock-in.
- You choose how **open or closed** your AI architecture is — but you never lose **control of your data**.
- Hybrid Manager provides a production-proven platform for deploying **trusted, governed AI** in the enterprise.

Learn more

- AI Factory Overview
- *Vector Engine (PGvector)*
- Pipelines and data preparation
- *Model Serving Reference Manual*
- Gen AI in Hybrid Manager

Security and governance in AI Factory

AI Factory helps you build Gen AI and data-driven applications on your infrastructure with clear guardrails. This page explains who is responsible for what, where governance is enforced, and how to apply least-privilege patterns across components.

Why security and governance matter

Gen AI systems touch sensitive data, call powerful models, and may execute actions. Proper controls reduce risk while keeping teams productive:

- Protect sensitive data at rest and in flight
- Limit who can deploy models, change retrieval rules, or publish assistants

- Provide auditability and observability for compliance
- Enforce least privilege when assistants call tools or external services

See also: *Sovereign AI Explained* .

Where enforcement happens

- Hybrid Manager (recommended): AI Factory is integrated into the Hybrid Manager control plane for governance and observability. Use the spoke for day-to-day operations:
- AI Factory in Hybrid Manager
- Gen AI in Hybrid Manager
- *Pipelines models reference*
- Outside Hybrid Manager (selected components): When you run components such as Pipelines and data preparation or *Vector Engine (PGvector)* outside Hybrid Manager, you own cluster lifecycle, access control, and observability.

Who is responsible

- Platform teams: Control cluster access, secrets, and network policy; provision GPUs for *Model Serving Reference Manual* .
- Data engineers: Govern sources, *Knowledge Bases* , and embedding freshness.
- Application/Gen AI builders: Define *Rulesets Manual* , *Retrievers Reference Manual* , *Tools Reference Manual* , and Assistants with least privilege.

How to apply governance in AI Factory

- Keep data in governed stores
- Use Postgres and object storage managed by your platform; control access with your standard policies.
- Use *Vector Engine (PGvector)* to keep embeddings close to data and reuse existing database controls.
- Govern retrieval and behavior
- Express retrieval scope and limits with Retrievers and Knowledge Bases.
- Enforce acceptable behavior with Rulesets; external calls should be explicit through Tools.
- Secure model endpoints
- Expose private endpoints via *Model Serving Reference Manual* with controlled access.
- Right-size GPU resources; see *How-To Update GPU Resources for an InferenceService* .
- Observe and audit
- In Hybrid Manager, use integrated observability to monitor model serving, Gen AI runs, and pipelines.
- Treat Threads, logs, and metrics as audit artifacts as required by your policy.

Prerequisites

- Hybrid Manager: A project with AI Factory enabled and entitlements for Gen AI, Pipelines, Vector search, and Model Serving. See the AI Factory spoke .
- Outside Hybrid Manager: Ensure an enterprise secrets store, network policy, and audit logging are in place for the components you run.

Use cases

- Regulated knowledge assistants: Use governed Knowledge Bases, constrained Retrievers, and Rulesets; run models via private endpoints.
- Document intelligence for PII: Keep content in Postgres/object storage, process via Pipelines, and restrict Tools to non-destructive operations.
- Internal automation: Gate write actions behind Tools with explicit scopes; review Threads and logs periodically.

Related topics

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Sovereign AI Explained*
- *Gen AI Builder*
- Pipelines and data preparation
- *Vector Engine (PGvector)*
- *Model Serving Reference Manual*

3.1 AI Factory Models

3.2 AI Factory Models

AI Factory Models provides comprehensive lifecycle management for AI models within EDB Postgres AI environments, delivering enterprise-grade governance and scalable inference capabilities through deep integration with Hybrid Manager infrastructure.

3.2.1 System Overview

AI Factory Models operates as a unified platform that bridges model governance and production deployment, ensuring organizational control over AI assets while enabling seamless integration across EDB PG AI capabilities. The system manages the complete model lifecycle from acquisition through retirement, providing consistent operational procedures and security controls.

Core Components

Model Library Centralized governance system that manages model image repositories, approval workflows, and metadata tracking. The library ensures only validated models reach production environments while maintaining comprehensive audit trails for compliance requirements.

Model Serving Infrastructure KServe-based deployment engine that transforms approved models into scalable inference endpoints within Kubernetes clusters managed by Hybrid Manager. The serving layer provides enterprise-grade operational characteristics including auto-scaling, health monitoring, and resource optimization.

Integration Framework Standardized interfaces that connect model inference capabilities with AI Accelerator Pipelines, Knowledge Bases, and Gen AI Builder applications, ensuring consistent model access patterns across all EDB PG AI workloads.

3.2.2 Strategic Benefits

Sovereign AI Implementation

Organizations achieve complete control over their AI infrastructure by deploying models within their own Kubernetes environments rather than relying on external API services. This approach ensures sensitive data and proprietary models remain within organizational boundaries while maintaining enterprise-grade operational capabilities.

Operational Consistency

Standardized deployment patterns and governance frameworks eliminate the complexity of managing diverse model types and frameworks. Teams benefit from unified operational procedures regardless of the underlying model architecture or intended use case.

Scalable Resource Management

Efficient GPU and compute resource allocation across multiple models optimizes infrastructure costs while maintaining performance guarantees. The system prevents resource underutilization through dynamic allocation strategies based on actual workload demands.

3.2.3 Architecture Integration

AI Factory Models integrates deeply with Hybrid Manager to provide comprehensive model lifecycle management within existing Kubernetes infrastructure. The system leverages Hybrid Manager's cluster orchestration capabilities while extending them with AI-specific governance and deployment features.

Hybrid Manager Dependencies

Asset Library Integration Model images, metadata, and version information are managed through Hybrid Manager's Asset Library, providing centralized storage and governance capabilities that extend beyond traditional container image management.

Kubernetes Orchestration Model serving workloads deploy as native Kubernetes resources through Hybrid Manager's cluster management capabilities, ensuring consistent operational behavior and resource allocation across all deployed models.

Security Framework Access control, network policies, and audit logging leverage Hybrid Manager's security infrastructure while adding model-specific governance controls and compliance features.

3.2.4 Operational Workflows

Model Lifecycle Management

Acquisition and Validation Models enter the system through automated synchronization from private registries or manual registration processes. Each model undergoes security scanning, performance validation, and compliance verification before approval for production use.

Deployment and Scaling Approved models deploy as InferenceServices that automatically scale based on demand patterns. The system optimizes resource allocation across multiple models while maintaining isolation and performance guarantees.

Monitoring and Maintenance Comprehensive observability provides visibility into model performance, resource utilization, and operational health. Automated alerting and recovery procedures minimize service disruptions while maintaining detailed audit trails.

Governance Framework

Approval Workflows Configurable approval processes ensure models meet organizational standards before production deployment. The system supports complex multi-stakeholder workflows while automating routine approvals for models meeting predefined criteria.

Policy Enforcement Security policies, resource constraints, and compliance requirements are enforced automatically across all model deployments. This ensures consistent application of organizational standards without manual intervention.

Audit and Compliance Detailed logging and reporting capabilities support regulatory compliance requirements while providing comprehensive visibility into model usage patterns and operational activities.

3.2.5 Learning Pathways

Foundation Understanding

Begin with core concepts to establish architectural understanding before progressing to practical implementation:

Conceptual Framework

1. *Model Serving Concepts* - Understanding inference deployment patterns
2. *Gen AI Architecture (Entities and Flows)* - System design and component relationships
3. *Model Library Explained In Depth* - Governance and lifecycle management principles

System Design

1. *Deployment Overview* - Production deployment patterns and considerations
2. *Observability for Model Serving* - Monitoring and operational visibility

Practical Implementation

Progress through hands-on implementation after establishing conceptual understanding:

Initial Setup

1. *Quickstart — Use an Inference Endpoint with Python* - Basic client integration
2. *How-To Setup GPU resources* - Infrastructure preparation for model workloads
3. *How-To Configure a ClusterServingRuntime* - Framework-specific deployment settings

Model Deployment

1. *How-To Create an InferenceService for NVIDIA NIM Container* - Basic model deployment procedures
2. *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe* - Specialized deployment patterns
3. *Accessing KServe endpoints (internal and external)* - Internal and external connectivity

Advanced Operations

1. *How-To Integrate Private Registry with Model Library* - Custom model source configuration
2. *How-To Define Repository Rules for Model Library* - Automated model discovery
3. *How-To Verify InferenceServices and GPU usage* - Deployment validation and troubleshooting

Reference Materials

Technical References

- *Model Serving Reference Manual* - Comprehensive deployment and configuration reference
- *Model Library Reference Manual* - Complete governance framework documentation
- *Gen AI Builder FAQ* - Common issues and implementation considerations

Integration Guides

- *AI Accelerator Pipelines* - Pipeline model integration
- *Knowledge Bases Reference Manual* - Application development with served models
- *Vector Engine (PGvector)* - Embedding model integration

3.2.6 Use Case Applications

Enterprise RAG Systems

Deploy embedding models for semantic search and retrieval augmented generation workflows that require consistent model versions across development and production environments.

Private LLM Deployment

Implement internal language model services that process sensitive data within organizational boundaries while maintaining enterprise-grade operational capabilities.

Multi-Modal AI Applications

Integrate vision, language, and embedding models into unified applications that require coordinated inference capabilities across different model types.

Hybrid Model Architectures

Combine private and external model services within applications that require both proprietary models for sensitive operations and external services for general capabilities.

3.2.7 Implementation Considerations

Infrastructure Requirements

Model serving requires substantial compute resources, particularly GPU infrastructure for large language models and vision workloads. Organizations should plan capacity based on anticipated model sizes and throughput requirements.

Network Architecture

Hybrid Manager cluster connectivity affects both model deployment and runtime access patterns. Consider network latency, bandwidth requirements, and security boundaries when designing model serving architectures.

Security Framework

Model governance extends beyond traditional container security to include AI-specific concerns such as model provenance, data privacy, and inference audit trails. Implement comprehensive security controls aligned with organizational AI governance policies.

Operational Complexity

Managing multiple models across different frameworks requires substantial operational expertise. Establish clear procedures for deployment, monitoring, and troubleshooting before scaling to production workloads.

AI Factory Models provides the foundation for enterprise AI operations through comprehensive model lifecycle management and scalable inference deployment within your controlled infrastructure environments.

Getting Started

Model Serving Concepts

Model Serving is a core capability of **EDB Postgres® AI** (EDB PG AI), enabling scalable, flexible, and high-performance serving of AI models on Kubernetes.

It powers:

- Gen AI applications
- Intelligent retrieval systems
- Advanced data pipelines

Model Serving is implemented using **KServe**, an open-source Kubernetes-native engine for standardized model inferencing. AI Factory integrates KServe with **Hybrid Manager** to provide enterprise-grade lifecycle management, security, and observability.

Key to Sovereign AI: Models run **on your Kubernetes clusters**, under your control, with full observability and governance.

Before you start

Prerequisites for understanding Model Serving:

- Familiarity with **Kubernetes basics** (pods, services, deployments)
- Understanding of **InferenceService** as a Kubernetes CRD
- Awareness of **Model Library** and how models are registered and deployed
- Understanding of **Sovereign AI principles** in EDB PG AI

Suggested path:

- *Fundamental AI Concepts*
- *Model Library Explained In Depth*

Why it matters

Model Serving enables your **AI models** and **Postgres data** to work together seamlessly — securely and scalably — under **Sovereign AI** principles:

- Deploy **open-source or commercial models** to your Kubernetes cluster.
- Serve Gen AI models for **Assistants**, **Knowledge Bases**, and **RAG** pipelines.
- Support **multi-modal retrieval** (text, image, hybrid search).

- Optimize performance with **GPU acceleration** and server-side batching.
 - Maintain full **observability**, **auditing**, and **governance** over model usage.
- See also: Hybrid Manager Model Serving integration in production environments.

Core concepts

InferenceService (via KServe)

At the core of Model Serving is the **InferenceService** — a Kubernetes-native resource that represents a deployed model.

It defines the end-to-end serving pipeline:

- **Predictor** — Runs the model server and handles inference.
- **Transformer** (optional) — Applies pre-processing or post-processing.
- **Explainer** (optional) — Provides model explainability outputs.

Predictor

The Predictor defines:

- **Model format** — PyTorch, TensorFlow, ONNX, Triton, etc.
- **Model location** — S3-compatible storage, OCI registry, PVC.
- **Resources** — CPU, memory, GPU.
- **Autoscaling** — Policies for elastic scaling, including scale-to-zero.

ServingRuntime / ClusterServingRuntime

Reusable runtime definitions for serving:

- **ServingRuntime** — Namespace-scoped.
- **ClusterServingRuntime** — Cluster-wide reusable runtimes.

Benefits:

- Tailor runtime settings to model type and hardware.
- Standardize runtime configurations across teams and projects.

How it works

Lifecycle flow

1. Register model image in **Model Library**.
2. Deploy model via **Model Serving** console or CLI → creates InferenceService.
3. AI Factory + KServe provision Kubernetes resources (pods, services).
4. Model is loaded into runtime container.
5. Kubernetes service endpoint is exposed.

6. Clients send HTTP/gRPC inference requests.
7. Requests may pass through Transformers and Explainers.
8. Inference response is returned.

Key features

- **Multi-framework support** — PyTorch, TensorFlow, ONNX, XGBoost, Triton, and more.
- **GPU acceleration** — Native NVIDIA GPU support.
- **Autoscaling** — Including scale-to-zero via Knative.
- **Observability** — Prometheus metrics, Kubernetes logging, and Hybrid Manager dashboards.
- **Batching** — Server-side batching for improved throughput.
- **Explainability** — Support for model explainability tooling.
- **Security and Sovereign AI** — Models run on your Kubernetes clusters under your control.

Patterns of use

Gen AI Builder

- Serve LLMs and multi-modal models powering **Assistants** and **Agents**.

Knowledge Bases

- Serve embedding and retrieval models used in:
 - **Knowledge Base indexing**
 - **RAG pipelines**

Custom applications

- Expose InferenceService endpoints to:
 - **Business applications**
 - **Microservices**
 - **ETL/ELT pipelines**

Hybrid + Sovereign AI alignment

- All models run **on your infrastructure** via Hybrid Manager KServe layer.
- You control:
 - Which models are deployed
 - Resource allocations
 - Deployment topology
 - Observability and auditing

Best practices

- Always deploy models through the **Model Library** → **Model Serving** flow to ensure governance.
- Monitor resource consumption — especially GPU utilization.
- Test **scale-to-zero** policies carefully before production use.
- Use **ServingRuntime** and **ClusterServingRuntime** templates for consistency.
- Tag and document **production models** clearly in Model Library.
- Audit InferenceService deployments regularly — critical for Sovereign AI.

In AI Factory

Model Serving powers multiple components in EDB PG AI:

Component	How it uses Model Serving
Gen AI Builder	Runs LLMs and specialized models for Assistants
Knowledge Bases	Serves embedding and retrieval models for RAG
Custom AI apps	Exposes InferenceService endpoints for business use cases

AI Factory manages Model Serving through:

- Integrated **Model Library** for image management.
- **GPU resource management** and scheduling.
- Centralized **observability** and logging.
- Seamless Hybrid Manager integration for lifecycle control.

Related topics

- [Model Library Explained In Depth](#)
- [How-To Deploy AI Models from Model Library](#)
- [How-To Create an InferenceService for NVIDIA NIM Container](#)
- [How-To Verify InferenceServices and GPU usage](#)
- [KServe Official Docs](#)
- [KServe GitHub Repository](#)

Next steps

- Explore available models in your **Model Library**.
- Deploy your first model using **Model Serving**.
- Monitor deployed models through Hybrid Manager observability dashboards.
- Explore advanced Model Serving capabilities:

- Multi-modal pipelines
- Transformers and Explainers
- Scale-to-zero policies

Model Serving gives you a powerful foundation for building intelligent, governed AI applications — securely and scalably — as part of your **Sovereign AI** strategy with **EDB PG AI**.

Gen AI Factory LLM Architecture (Library and Serving Overview)

Understanding EDB AI Factory’s Model Library and LLM Serving Components

This document provides a conceptual framework for understanding how Model Library and Model Serving components work together within AI Factory to deliver enterprise-grade Large Language Model (LLM) inference capabilities. Organizations implementing AI Factory require this architectural understanding to make informed decisions about LLM deployment strategies, resource allocation, and governance frameworks that align with their operational requirements and compliance obligations.

Purpose and Scope

AI Factory’s LLM serving architecture enables organizations to deploy and manage Large Language Models at scale while maintaining governance, security, and operational control. The system addresses the critical gap between experimental LLM development and production-ready inference services, where traditional approaches often fail to meet enterprise requirements for reliability, security, and compliance. Organizations benefit from reduced operational complexity, improved GPU resource utilization, and consistent policy enforcement across their entire LLM lifecycle. The architecture bridges the gap between LLM development and production deployment through two primary components: the Model Library for curation and the Model Serving infrastructure for runtime operations.

Architectural Principles

Separation of Concerns

The architecture distinguishes between LLM storage/curation (Model Library) and LLM execution (Model Serving), enabling independent scaling and management of each concern. This separation allows platform teams to optimize storage strategies without impacting running inference services, while AI teams can focus on LLM performance without worrying about underlying infrastructure complexity. Organizations experience improved system stability because failures in one component don’t cascade to others, and teams can implement changes to curation policies or serving configurations independently based on their specific operational cycles.

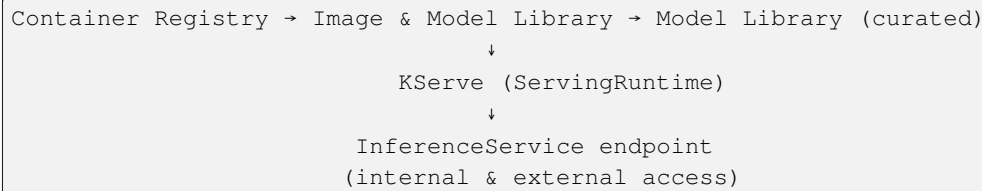
Governance Integration

Security and compliance controls are embedded throughout the LLM pipeline rather than added as afterthoughts, ensuring consistent policy enforcement from image ingestion to production inference. This approach eliminates the common problem of governance gaps that occur when security measures are applied retroactively to existing systems. Platform administrators benefit from centralized policy management that automatically applies to all LLMs, while compliance teams gain comprehensive audit trails and enforcement mechanisms that meet regulatory requirements without manual intervention.

Resource Optimization

GPU resources and compute infrastructure are abstracted from individual LLM deployments, allowing efficient resource sharing and dynamic allocation based on workload demands. This abstraction prevents the common scenario where expensive GPU resources remain underutilized because they're statically allocated to specific LLMs. Organizations achieve significant cost savings through improved resource utilization while maintaining performance guarantees, and development teams benefit from simplified deployment processes that don't require detailed infrastructure knowledge.

Control and Data Flow



Flow Components

1. **LLM Image Synchronization:** LLM container images are synchronized from registries into the Image & Model Library based on repository rules that define which repositories, tags, and LLM types to ingest
2. **Curation Process:** Model Library exposes a curated subset of those LLM images for serving through governed selection processes
3. **Deployment Configuration:** Operators select a ServingRuntime optimized for LLM frameworks and deploy an InferenceService with the chosen LLM image
4. **Access Management:** Applications invoke LLMs via internal DNS for cluster communication or external portal endpoints with access keys, depending on network and security requirements

Core Components

Image & Model Library

The foundational registry system that synchronizes LLM container images from external sources and maintains a centralized catalog of available language models. This component serves as the authoritative source for all LLM images within the organization, eliminating the chaos that typically results from teams maintaining separate, disconnected model repositories. Platform teams gain complete visibility into LLM inventory across the organization, while AI teams benefit from streamlined access to approved models without navigating complex approval processes for each deployment.

The Image & Model Library continuously monitors configured external registries and automatically ingests new LLM versions based on defined criteria, ensuring teams always have access to the latest approved language models without manual intervention. This automation reduces the operational burden on platform teams while ensuring consistent application of organizational policies across all ingested content.

Key Functions:

- Automated synchronization from configured container registries reduces manual overhead while ensuring consistency across LLM versions and sources

- Repository rule enforcement for selective LLM ingestion prevents unauthorized or non-compliant models from entering the organization's deployment pipeline
- Metadata management and version tracking provides complete lineage information that supports both operational troubleshooting and compliance reporting requirements
- Security scanning and vulnerability assessment automatically identifies potential risks before LLMs reach production environments

Integration Points:

- *How-To Define Repository Rules for Model Library*
- *How-To Integrate Private Registry with Model Library*
- *How-To Manage Repository and Image Tag Metadata*

Model Library (Curated View)

A governed subset of the Image & Model Library that presents only approved LLMs for production deployment, implementing organizational policies and compliance requirements. This curated view transforms the comprehensive LLM catalog into a focused collection of language models that meet specific organizational standards for security, performance, and compliance. Development teams benefit from simplified LLM selection because they can choose from any model in the curated library with confidence that it meets all organizational requirements.

The curation process acts as a quality gate that prevents problematic LLMs from reaching production while maintaining development velocity for teams working with approved models. Platform administrators can implement sophisticated approval workflows that automatically evaluate LLMs against multiple criteria, reducing manual review burden while maintaining strict quality standards.

Governance Controls:

- Approval workflows for LLM promotion ensure that only models meeting organizational standards reach production environments, reducing risk while maintaining development agility
- Security policy enforcement automatically applies consistent security standards across all LLMs without requiring individual team expertise in security best practices
- Performance benchmarking requirements establish minimum performance thresholds that ensure production LLMs meet operational requirements before deployment
- Compliance validation checks automatically verify that LLMs meet industry-specific regulatory requirements, reducing compliance risk and audit preparation overhead

Related Documentation:

- *Model Library Reference Manual*
- *Model Library Explained In Depth*

LLM Serving Infrastructure

KServe-based deployment system that transforms approved LLM images into scalable inference endpoints with enterprise-grade operational characteristics. This infrastructure handles the complex orchestration required to deploy Large Language Models reliably at scale, including health monitoring, auto-scaling, and GPU resource management. Development teams benefit from simplified deployment processes that abstract away infrastructure complexity, while operations teams gain comprehensive monitoring and management capabilities for all deployed LLMs.

The serving infrastructure provides consistent operational characteristics across different LLM types and frameworks, ensuring that teams can apply standard operational procedures regardless of the underlying model architecture. This consistency reduces operational complexity while improving reliability through standardized monitoring, logging, and recovery procedures.

Core Resources:

- **ServingRuntime:** LLM-specific execution environments (vLLM, NIM, TensorRT-LLM) provide optimized runtime characteristics for different language model types while abstracting framework-specific configuration complexity from deployment teams
- **InferenceService:** Production LLM endpoints with auto-scaling capabilities automatically adjust resource allocation based on demand patterns, ensuring optimal performance while minimizing costs
- **Resource Management:** GPU allocation and compute optimization systems ensure efficient utilization of expensive hardware resources across multiple LLMs and workloads

Implementation Guides:

- *Model Serving Concepts*
- *How-To Configure a ClusterServingRuntime*
- *How-To Create an InferenceService for NVIDIA NIM Container*

Access Management

Multi-layered access control system that provides both internal cluster communication and external API access while maintaining security boundaries. This system addresses the complex security requirements of production LLM systems where models may need to serve both internal microservices and external applications with different security profiles. Platform administrators benefit from centralized access control that consistently applies security policies across all access patterns, while development teams gain flexible access options that support diverse integration requirements.

The access management system provides granular control over who can access which LLMs under what circumstances, supporting complex organizational requirements while maintaining operational simplicity. This approach eliminates common security gaps that occur when access controls are applied inconsistently across different access methods.

Access Patterns:

- **Internal Access:** Cluster-local DNS for service-to-service communication provides high-performance, low-latency access for applications running within the same infrastructure while maintaining network-level security isolation
- **External Access:** Portal-based endpoints with authentication and authorization enable secure integration with external applications and third-party systems while maintaining comprehensive audit trails and access control

Configuration Details:

- *Accessing KServe endpoints (internal and external)*
-

System Flow**LLM Ingestion Pipeline**

LLM container images flow from external registries into the Image & Model Library through automated synchronization processes governed by repository rules. These rules determine which repositories, namespaces, and tags to monitor and ingest, providing organizations with precise control over their LLM pipeline while automating routine synchronization tasks. Platform teams benefit from reduced manual overhead in managing LLM updates, while maintaining strict control over which models enter their environment. AI teams gain access to approved language models more quickly because the automated pipeline eliminates manual approval bottlenecks for models that meet predefined criteria.

The ingestion pipeline provides comprehensive logging and monitoring that enables platform teams to track LLM provenance and identify potential issues before they impact production systems. This visibility supports both operational troubleshooting and compliance reporting requirements.

Curation Process

The Model Library applies organizational governance policies to filter the complete LLM catalog, presenting only approved models for deployment. This curation process includes security scanning, performance validation, and compliance verification that automatically evaluates LLMs against organizational standards without requiring manual review for every model version. The process reduces time-to-production for compliant LLMs while maintaining strict quality gates that prevent problematic models from reaching production environments.

Organizations benefit from consistent application of governance policies across all LLMs, regardless of their source or development team. This consistency reduces compliance risk while enabling teams to move quickly with models that meet organizational standards.

Deployment Workflow

Approved LLMs are deployed through the Model Serving infrastructure, where `ServingRuntime` configurations define the execution environment and `InferenceService` specifications determine scaling behavior and resource requirements. This workflow abstracts the complexity of production deployment while providing comprehensive control over operational characteristics. Development teams benefit from simplified deployment processes that don't require deep infrastructure expertise, while platform teams maintain control over resource allocation and operational policies.

The deployment workflow provides automated validation and rollback capabilities that reduce deployment risk while maintaining development velocity. Teams can deploy LLMs with confidence because the system automatically verifies operational readiness before directing production traffic to new deployments.

Runtime Operations

Production LLM inference endpoints operate under continuous monitoring with automated scaling, health checking, and performance optimization. The system maintains operational metrics and provides observability into model performance and resource utilization, enabling proactive management of production systems. Operations teams benefit from comprehensive visibility into system health and performance, while development teams gain insights into LLM behavior under production conditions.

The runtime operations framework provides automated recovery capabilities that minimize service disruptions while maintaining detailed audit trails for troubleshooting and compliance purposes. This automation reduces operational burden while improving system reliability through consistent application of operational best practices.

Resource Management

GPU Infrastructure for LLMs

The architecture abstracts GPU resources from individual LLM deployments, enabling efficient resource sharing and dynamic allocation based on demand patterns. This abstraction addresses the common challenge of GPU underutilization that occurs when resources are statically allocated to specific LLMs, regardless of actual demand. Organizations achieve significant cost savings through improved resource utilization while maintaining performance guarantees for production workloads.

Platform teams benefit from centralized resource management that optimizes utilization across all deployed LLMs, while development teams gain access to GPU resources without needing to understand complex allocation strategies. The system automatically handles resource scheduling and allocation based on workload characteristics and organizational priorities.

Setup Requirements:

- [*How-To Setup GPU resources*](#)
- [*How-To Update GPU Resources for an InferenceService*](#)

LLM Scaling Strategies

Auto-scaling capabilities adjust compute resources based on traffic patterns and performance requirements, ensuring cost-effective operations while maintaining service levels. The scaling system learns from historical patterns and proactively adjusts resources to meet demand, reducing both costs and latency for end users. Organizations benefit from optimized resource costs without compromising performance, while development teams can focus on LLM development without worrying about operational scaling concerns.

The scaling strategies support both predictable workload patterns and sudden traffic spikes, ensuring consistent performance across diverse usage scenarios. This flexibility enables organizations to serve LLMs efficiently regardless of demand variability.

Performance Optimization for LLMs

The system incorporates LLM-specific optimizations including batch processing, quantization, and tensor parallelism to maximize throughput and minimize latency. These optimizations are applied automatically based on model characteristics and workload patterns, ensuring optimal performance without requiring specialized expertise from development teams. Organizations benefit from improved LLM performance and reduced infrastructure costs through efficient resource utilization.

Performance optimizations are continuously evaluated and adjusted based on actual workload characteristics, ensuring that optimization strategies evolve with changing requirements and model characteristics. This adaptive approach maintains optimal performance as workloads and LLMs evolve over time.

Security and Governance

Access Control Framework

Multi-tiered security model that enforces authentication and authorization at multiple levels, from registry access through production inference endpoints. This framework addresses the complex security requirements of production LLM systems where different users and applications require different levels of access to various models and capabilities. Platform administrators benefit from centralized policy management that consistently applies security controls across all system components, while maintaining flexibility to support diverse access requirements.

The access control framework provides comprehensive audit trails that support compliance reporting and security monitoring, enabling organizations to maintain security visibility across their entire LLM infrastructure. This visibility supports both operational security monitoring and compliance reporting requirements.

Compliance Integration

Built-in compliance controls ensure that LLM deployments meet organizational and regulatory requirements throughout the entire lifecycle. These controls are integrated into every stage of the model pipeline, from ingestion through production deployment, ensuring consistent compliance without impacting development velocity. Organizations benefit from automated compliance verification that reduces manual audit overhead while maintaining comprehensive compliance coverage.

The compliance integration provides detailed documentation and audit trails that support regulatory reporting requirements while enabling organizations to demonstrate compliance to auditors and regulators. This documentation reduces compliance burden while providing comprehensive evidence of control effectiveness.

Audit and Monitoring

Comprehensive logging and monitoring capabilities provide visibility into system operations, LLM performance, and security events. This monitoring framework enables proactive identification of issues before they impact production systems while providing detailed forensic capabilities for troubleshooting and compliance investigations. Operations teams benefit from comprehensive system visibility that supports both day-to-day operations and incident response, while compliance teams gain detailed audit trails that support regulatory requirements.

The monitoring system provides both real-time alerting for immediate issues and historical analysis capabilities that support capacity planning and performance optimization. This dual approach ensures both operational responsiveness and strategic planning capabilities.

Monitoring Setup:

- *Observability for Model Serving*
 - *How-To Monitor deployed models with KServe*
-

Integration Patterns

Development Workflow

The system integrates with existing development processes through SDK support and API compatibility, enabling seamless integration with application development lifecycles. This integration reduces friction for development teams while maintaining the benefits of centralized LLM management and governance. Development teams can incorporate LLM capabilities into their applications using familiar tools and patterns, while benefiting from enterprise-grade operational capabilities provided by the platform.

Integration with development workflows supports both experimental development and production deployment through consistent interfaces that scale from prototype to production without requiring significant changes to application code or deployment processes.

Getting Started:

- *Quickstart — Use an Inference Endpoint with Python*
- *How-To Deploy AI Models from Model Library*

API Compatibility

Standard API interfaces ensure compatibility with existing LLM toolchains while providing enterprise-grade capabilities for production workloads. This compatibility enables organizations to leverage existing investments in tools and processes while gaining the benefits of centralized LLM management and governance. Development teams can use familiar APIs and tools while benefiting from improved reliability, security, and operational capabilities provided by the platform.

API compatibility extends to both LLM training workflows and application integration patterns, ensuring that teams can adopt the platform without significant changes to existing processes and tools.

Advanced Capabilities

Multi-LLM Framework Support

The architecture accommodates various LLM frameworks and runtime environments, enabling organizations to deploy diverse language model types within a unified infrastructure. This flexibility prevents vendor lock-in while enabling

teams to choose optimal frameworks for specific use cases without fragmenting operational processes. Organizations benefit from consistent operational procedures across different LLM types while maintaining flexibility to adopt new frameworks and technologies as they mature.

Multi-framework support includes optimized runtime configurations for different LLM types that ensure optimal performance characteristics regardless of underlying framework technology. This optimization reduces the expertise required from development teams while ensuring optimal resource utilization and performance.

Specialized Deployments:

- *How-To Deploy an NVIDIA NIM Microservices available on [build.nvidia.com/container with KServe](https://build.nvidia.com/container-with-KServe)*

Custom Runtime Configuration

Organizations can define custom `ServingRuntime` configurations to support specialized LLM requirements or optimization strategies. This customization capability enables support for unique organizational requirements while maintaining the benefits of standardized operational processes. Platform teams can create specialized runtime configurations that optimize for specific LLM characteristics or business requirements while ensuring consistent operational behavior across all deployments.

Custom runtime configurations support both standard optimization patterns and organization-specific requirements that may not be addressed by default configurations. This flexibility ensures that the platform can adapt to diverse organizational needs while maintaining operational consistency.

Next Steps

Understanding this architectural overview provides the foundation for implementing LLM serving capabilities within your organization. The referenced implementation guides provide detailed instructions for specific configuration tasks and operational procedures that translate these architectural concepts into working systems. Organizations should begin with the foundational components and progress through advanced capabilities based on their specific operational maturity and business requirements.

For hands-on experience with the system, begin with the quickstart guides and progress through the detailed configuration documentation based on your specific deployment requirements. This progression ensures that implementation efforts align with architectural principles while building practical expertise with system capabilities and operational procedures.

Deployment Overview

AI Factory makes it easy to deploy your AI models as scalable, production-ready inference services. The platform uses **KServe** as the model serving engine, operating within the Hybrid Manager (HM) Kubernetes infrastructure.

This page explains the general flow of model deployment and links to key how-to guides for hands-on instructions.

Deployment flow overview

- You deploy models by creating **KServe InferenceServices** in your HM project.
 - AI Factory provides **GPU-enabled Kubernetes infrastructure** to run these services.
 - You can deploy supported **NVIDIA NIM Microservices** or other compatible models.
 - The **Model Library** helps you discover and manage model images.
 - Applications access model endpoints over **HTTP** or **gRPC** APIs.
-

Deployment components

KServe InferenceService

Each model is deployed via a Kubernetes-native `InferenceService` object:

- Manages lifecycle of the model server pods.
- Handles scaling, health checks, and routing.
- Exposes a network endpoint for model consumption.

ClusterServingRuntime

Advanced users can also configure `ClusterServingRuntime` resources to customize runtime environments for their models.

Where to start

If you're ready to deploy models, follow these guides:

- [*How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.comcontainer with KServe*](#)
 - [*How-To Create an InferenceService for NVIDIA NIM Container*](#)
 - [*How-To Configure a ClusterServingRuntime*](#)
 - [*How-To Monitor deployed models with KServe*](#)
 - [*How-To Update GPU Resources for an InferenceService*](#)
-

Hybrid Manager integration

Model Serving runs on **Hybrid Manager (HM)** Kubernetes clusters. For more on Hybrid Manager and GPU setup:

- [*How-To Setup GPU resources*](#)
 - [*How-To Verify InferenceServices and GPU usage*](#)
-

Best practices

- Use the Model Library to select supported models.
 - Verify that your cluster has sufficient GPU resources.
 - Monitor deployed models to ensure performance and availability.
 - Use `ClusterServingRuntime` where advanced customization is needed.
-

Next steps

- Explore our *How-To Deploy AI Models from Model Library*
 - Review supported models in the *Supported Models Index*
 - Learn about *Observability for Model Serving*
-

By following this deployment flow, you can run AI models in production with full observability and scale — directly integrated with the broader AI Factory and Hybrid Manager ecosystem.

AI Factory Model Use Cases and Personas

This guide outlines practical implementation scenarios for AI Factory Models across different organizational personas and use cases. Each scenario provides specific implementation pathways aligned with role-based requirements.

Persona-Based Implementation Scenarios

Platform Engineers

Platform Engineers focus on infrastructure provisioning, system reliability, and operational excellence across AI model deployments.

Infrastructure Management Scenario

Objective: Establish reliable, scalable model serving infrastructure that supports multiple development teams and production workloads.

Key Requirements:

- Multi-tenant resource isolation for different teams and projects
- Automated scaling based on inference demand patterns
- Comprehensive monitoring and alerting for proactive issue resolution
- Security controls that align with organizational policies

Implementation Approach:

1. **Infrastructure Foundation**

- Configure GPU-enabled Kubernetes nodes with appropriate taints and labels
- Implement network policies for secure multi-tenant isolation
- Establish resource quotas and limits aligned with organizational capacity planning

2. **Operational Framework**

- Deploy monitoring stack integration for inference service observability
- Configure automated backup procedures for model metadata and configurations
- Implement disaster recovery procedures for critical model serving workloads

3. **Governance Integration**

- Establish approval workflows for production model deployments
- Configure security scanning pipelines for model image validation
- Implement audit logging for compliance and operational forensics

Reference Documentation:

- *How-To Setup GPU resources*
 - *Observability for Model Serving*
 - *How-To Verify InferenceServices and GPU usage*
-

AI Engineers

AI Engineers require streamlined model deployment workflows that abstract infrastructure complexity while providing comprehensive control over model performance characteristics.

Model Deployment Optimization Scenario

Objective: Deploy and optimize large language models for production inference with minimal operational overhead.

Key Requirements:

- Framework-agnostic deployment procedures that support diverse model types
- Performance optimization capabilities including quantization and batch processing
- A/B testing frameworks for comparing model variants
- Integration with existing AI pipelines and workflows

Implementation Strategy:**1. Model Preparation**

- Register model images in the Asset Library with comprehensive metadata
- Configure ServingRuntime optimizations for specific model frameworks
- Establish performance baselines through systematic benchmarking

2. Deployment Configuration

- Create InferenceServices with resource allocation aligned to model requirements
- Implement auto-scaling policies based on inference demand patterns
- Configure canary deployment strategies for safe production updates

3. Performance Monitoring

- Establish model-specific metrics for latency, throughput, and accuracy tracking
- Implement automated performance regression detection
- Configure alerting for model performance degradation

Reference Documentation:

- *How-To Configure a ClusterServingRuntime*
 - *How-To Create an InferenceService for NVIDIA NIM Container*
 - *How-To Monitor deployed models with KServe*
-

Application Developers

Application Developers integrate model inference capabilities into business applications while maintaining development velocity and operational simplicity.

RAG Application Development Scenario

Objective: Build retrieval-augmented generation applications that leverage private embedding models and language models within organizational infrastructure.

Key Requirements:

- Standardized API interfaces compatible with existing application frameworks
- Consistent model versions across development and production environments
- Low-latency inference for interactive user experiences
- Integration with vector databases and knowledge management systems

Development Workflow:

1. **Model Integration**

- Identify approved embedding and language models through the Model Library
- Configure application clients using standardized OpenAI-compatible endpoints
- Implement error handling and fallback strategies for model availability

2. **Performance Optimization**

- Implement caching strategies for frequently requested embeddings
- Configure batch processing for bulk document embedding operations
- Optimize request patterns to minimize inference latency

3. **Production Deployment**

- Establish environment-specific model endpoint configurations
- Implement monitoring for application-specific model usage patterns
- Configure automated failover between model instances

Reference Documentation:

- [*Quickstart — Use an Inference Endpoint with Python*](#)
 - [*Accessing KServe endpoints \(internal and external\)*](#)
 - Gen AI Builder Integration
-

DevOps Engineers

DevOps Engineers focus on CI/CD integration, deployment automation, and operational reliability for model serving workloads.

Automated Model Deployment Pipeline Scenario

Objective: Implement continuous deployment pipelines that automatically update model serving infrastructure based on model registry changes.

Key Requirements:

- Automated model validation and deployment workflows
- Integration with existing CI/CD infrastructure and approval processes
- Rollback capabilities for failed deployments or performance regressions
- Comprehensive logging and audit trails for deployment activities

Pipeline Implementation:

1. **Automation Framework**

- Configure webhook triggers for model registry updates
- Implement automated validation pipelines for model compatibility testing
- Establish deployment gates based on performance and security criteria

2. **Deployment Orchestration**

- Create infrastructure-as-code templates for InferenceService deployments
- Implement blue-green deployment strategies for zero-downtime updates
- Configure automated rollback procedures based on health check failures

3. **Monitoring Integration**

- Establish deployment success/failure metrics and alerting
- Implement performance comparison dashboards for pre/post deployment analysis
- Configure audit logging for deployment activities and approval workflows

Reference Documentation:

- *How-To Deploy AI Models from Model Library*
 - *How-To Integrate Private Registry with Model Library*
 - *How-To Define Repository Rules for Model Library*
-

Security Engineers

Security Engineers implement comprehensive security controls for model serving infrastructure while ensuring compliance with organizational and regulatory requirements.

Secure Model Serving Architecture Scenario

Objective: Establish security controls that protect model assets and inference data throughout the model serving life-cycle.

Key Requirements:

- Zero-trust network architecture for model inference traffic
- Comprehensive audit logging for model access and usage patterns

- Vulnerability management for model images and serving infrastructure
- Data privacy controls for inference requests and responses

Security Implementation:

1. Access Control Framework

- Implement role-based access control for model registration and deployment operations
- Configure network policies for secure multi-tenant model isolation
- Establish API authentication and authorization for inference endpoints

2. Vulnerability Management

- Configure automated security scanning for model images during registration
- Implement vulnerability remediation workflows for identified security issues
- Establish security approval gates for production model deployments

3. Audit and Compliance

- Configure comprehensive audit logging for all model operations
- Implement data retention policies aligned with regulatory requirements
- Establish compliance reporting capabilities for security assessments

Reference Documentation:

- *Model Library Reference Manual*
- *Accessing KServe endpoints (internal and external)*
- *Observability for Model Serving*

Cross-Functional Use Cases

Enterprise Sovereign AI Implementation

Scenario: Large enterprise implementing comprehensive AI governance while maintaining operational efficiency across multiple business units.

Stakeholders: Platform Engineers, Security Engineers, Compliance Teams

Implementation Requirements:

- Multi-tenant model serving infrastructure supporting diverse business requirements
- Comprehensive governance framework ensuring regulatory compliance
- Integration with existing enterprise identity and access management systems
- Performance optimization across different model types and usage patterns

Solution Architecture:

1. Infrastructure Layer

- Deploy Hybrid Manager across multiple Kubernetes clusters for geographic distribution
- Implement federated model registry with synchronized governance policies
- Configure cross-cluster networking for model sharing and load balancing

2. Governance Integration

- Establish enterprise-wide approval workflows with business unit delegation

- Implement automated compliance reporting for regulatory requirements
- Configure security policies aligned with enterprise risk management frameworks

3. Operational Excellence

- Deploy comprehensive monitoring across all model serving infrastructure
- Implement automated capacity planning based on historical usage patterns
- Establish incident response procedures for model serving disruptions

Hybrid Cloud Model Deployment

Scenario: Organization deploying models across on-premises and cloud infrastructure while maintaining consistent operational procedures.

Stakeholders: Platform Engineers, DevOps Engineers, Network Engineers

Implementation Considerations:

- Network connectivity and latency optimization between deployment environments
- Consistent security policies across different infrastructure providers
- Data locality requirements for model inference and training workloads
- Disaster recovery capabilities spanning multiple infrastructure tiers

Architecture Components:

- Federated Hybrid Manager deployment across cloud and on-premises infrastructure
- Unified model registry with environment-specific deployment configurations
- Cross-environment monitoring and alerting integration
- Consistent backup and disaster recovery procedures

Implementation Considerations

Resource Planning

Model serving infrastructure requires substantial computational resources, particularly for large language models and high-throughput inference workloads. Organizations should establish capacity planning procedures based on anticipated model sizes and usage patterns.

Resource Allocation Guidelines:

- GPU memory requirements scale directly with model parameter counts
- CPU allocation affects preprocessing and postprocessing performance
- Network bandwidth impacts high-throughput inference scenarios
- Storage performance affects model loading and checkpoint operations

Operational Complexity

Managing diverse model types across different frameworks requires substantial operational expertise. Establish comprehensive training programs and operational procedures before scaling to production workloads.

Operational Best Practices:

- Implement standardized deployment procedures across different model types

- Establish clear escalation procedures for operational issues
- Maintain comprehensive documentation for troubleshooting and maintenance
- Regular operational reviews to identify improvement opportunities

Performance Optimization

Model inference performance depends on multiple factors including hardware configuration, model characteristics, and application usage patterns. Implement systematic performance testing and optimization procedures.

Optimization Strategies:

- Establish performance baselines for all deployed models
- Implement automated performance regression detection
- Configure resource allocation based on actual usage patterns
- Regular performance reviews and optimization cycles

These use cases provide concrete implementation pathways for different organizational roles while highlighting the comprehensive capabilities of AI Factory Models across diverse deployment scenarios.

Model Serving FAQ

Here are answers to common questions about using Model Serving with AI Factory.

What models are currently supported?

AI Factory Model Serving (version 1.2) supports deploying models packaged as **NVIDIA NIM Microservices**. Support for deploying additional model types and formats will be expanded in future releases.

What Kubernetes resources are used for serving models?

Models are served using [KServe InferenceServices](#), which manage the lifecycle of model-serving pods. Supporting KServe resources such as `ServingRuntimes` and related GPU-configured Kubernetes nodes are also used.

What hardware is required?

NIM-based model serving requires **GPU-enabled Kubernetes nodes**.

Refer to the [How-To Update GPU Resources for an InferenceService](#) for supported configurations and node recommendations by cloud provider.

Can I deploy custom models?

At this time, AI Factory Model Serving is optimized for **NIM-based containers**. Support for deploying arbitrary models via KServe is planned, but requires manual configuration outside of the current AI Factory streamlined flow.

How is scaling handled?

KServe InferenceServices use **Kubernetes-native scaling**, with auto-scaling supported for compatible runtimes. Scaling behavior can be configured in the InferenceService specification (replicas, resources).

How do I monitor running models?

You can monitor deployed models using:

- **KServe InferenceService status**
- Kubernetes monitoring tools (Prometheus, Grafana, etc.)
- *How-To Monitor deployed models with KServe*

Where can I learn more about the model serving architecture?

See:

- *Model Serving Concepts*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*

Have more questions? Please contact your AI Factory administrator or consult AI Factory Support Resources .

Manuals

Model Serving Reference Manual

Model Serving transforms validated AI models into production-ready inference endpoints that integrate seamlessly with EDB Postgres AI capabilities. The system deploys containerized models as scalable services within Kubernetes environments managed by Hybrid Manager.

Prerequisites: Model Serving requires Hybrid Manager installation and *Asset Library configuration* for model image management. All inference services operate within your controlled Kubernetes environment.

System Architecture

KServe Foundation

Model Serving utilizes KServe as the underlying orchestration engine, providing enterprise-grade capabilities for model deployment, scaling, and lifecycle management. KServe abstracts the complexity of containerized model deployment while maintaining comprehensive control over inference operations.

Core Components:

- **ServingRuntime:** Framework-specific execution environments optimized for different model types
- **InferenceService:** Production endpoints with automatic scaling and health management
- **Resource Management:** GPU allocation and compute optimization across deployed models

Integration Framework

The serving infrastructure operates as a bridge between the Model Library's governed model catalog and EDB PG AI's operational capabilities. Models approved through governance workflows become available for deployment as inference services that support various AI workloads.

Supported Model Types:

- Large Language Models (LLMs) for text generation and completion
- Embedding models for vector search and retrieval augmented generation
- Vision models including CLIP, OCR, and image embedding capabilities

- Reranking models for search result optimization

Operational Characteristics

Deployment Patterns

InferenceServices support multiple deployment strategies based on organizational requirements and model characteristics. The system accommodates both high-availability production deployments and development environments with appropriate resource allocation and scaling behaviors.

Deployment Configurations:

- **Single-model deployments** for dedicated inference workloads
- **Multi-model serving** for resource efficiency with compatible models
- **Canary deployments** for safe production updates with gradual traffic shifting
- **A/B testing configurations** for model performance comparison

Scaling Mechanisms

Automatic scaling adjusts compute resources based on inference demand, optimizing cost while maintaining service level objectives. The system supports both horizontal pod scaling and vertical resource adjustment based on workload characteristics.

Scaling Factors:

- Request concurrency and queue depth
- Model-specific latency requirements
- GPU memory utilization patterns
- Custom metrics for specialized workloads

Resource Management

GPU resources are managed through centralized allocation strategies that optimize utilization across multiple models. The system supports various GPU configurations and provides isolation mechanisms to prevent resource contention.

Resource Considerations:

- GPU memory requirements for model loading and inference
- CPU allocation for preprocessing and postprocessing operations
- Network bandwidth for high-throughput inference workloads
- Storage requirements for model artifacts and temporary data

Access and Integration

Endpoint Configuration

InferenceServices expose standardized API endpoints that provide compatibility with common AI frameworks and applications. The system supports both internal cluster communication and external access through secured gateway configurations.

Internal Access Patterns:

- Cluster-local DNS resolution for service-to-service communication

- Network policy enforcement for security isolation
- Service mesh integration for advanced traffic management

External Access Configuration:

- Portal-based endpoints with authentication and authorization
- API key management for secure external integration
- Rate limiting and quota enforcement capabilities

API Compatibility

Standard endpoint paths provide compatibility with OpenAI-compatible clients and custom applications, enabling seamless integration with existing AI workflows and toolchains.

Endpoint Specifications:

Chat completions (LLMs):	<base>/v1/chat/completions
Embeddings:	<base>/v1/embeddings
Reranking:	<base>/v1/ranking

EDB PG AI Integration

Model Serving provides inference capabilities for all EDB PG AI components through consistent interfaces that abstract deployment complexity while maintaining operational control.

Integration Points:

- **AI Accelerator Pipelines** leverage embedding and preprocessing models for data transformation workflows
- **Knowledge Bases** utilize embedding models for vector search and semantic retrieval operations
- **Gen AI Builder** applications access LLMs for conversational AI and text generation capabilities

Configuration Framework

ServingRuntime Management

ServingRuntime configurations define the execution environment for different model frameworks, providing optimized performance characteristics while abstracting framework-specific complexity from deployment operations.

Runtime Optimization Features:

- Framework-specific optimizations for model loading and inference
- GPU kernel optimization for improved throughput
- Memory management strategies for large model deployment
- Batch processing capabilities for improved resource utilization

How-To Configure a ClusterServingRuntime

Resource Allocation

Proper resource configuration ensures optimal performance while avoiding resource contention and out-of-memory conditions. Organizations must balance performance requirements with infrastructure costs when defining resource specifications.

Critical Configuration Areas:

- CPU and memory requests/limits for container scheduling
- GPU allocation and sharing strategies
- Network and storage performance requirements
- Health probe and timeout configurations

How-To Update GPU Resources for an InferenceService

Observability Configuration

Comprehensive monitoring enables proactive management of inference services while providing visibility into performance characteristics and resource utilization patterns.

Monitoring Capabilities:

- Request latency and throughput metrics
- Resource utilization tracking across CPU, memory, and GPU
- Error rate monitoring with detailed failure analysis
- Custom metrics for application-specific requirements

Observability for Model Serving

Operational Considerations

Performance Optimization

Model Serving performance depends on multiple factors including model characteristics, hardware configuration, and traffic patterns. Organizations should establish baseline performance metrics and continuously optimize based on production usage patterns.

Optimization Strategies:

- Batch size tuning for optimal throughput versus latency trade-offs
- Quantization techniques for memory efficiency without significant accuracy loss
- Caching strategies for frequently accessed models and predictions
- Request routing optimization for multi-model deployments

Security Framework

Production model serving requires comprehensive security controls that protect both model assets and inference data while maintaining operational efficiency.

Security Controls:

- Network isolation through Kubernetes network policies
- Authentication and authorization for API access
- Model image security scanning and vulnerability management
- Audit logging for compliance and forensic analysis

Scalability Limitations

Understanding system limitations enables appropriate capacity planning and architectural decisions for large-scale deployments.

Scaling Constraints:

- GPU memory limits model size and batch processing capabilities
- Network bandwidth affects high-throughput inference workloads
- Storage I/O performance impacts model loading and checkpoint operations
- Kubernetes cluster limits affect maximum concurrent deployments

Troubleshooting Framework

Common Issues

Model Serving deployments may encounter various operational issues that require systematic troubleshooting approaches.

Typical Problems:

- **HTTP 404 errors** often indicate incorrect endpoint paths or service misconfiguration
- **Resource allocation failures** typically result from insufficient GPU or memory availability
- **Model loading timeouts** may require adjusted health probe configurations
- **Performance degradation** often indicates resource contention or suboptimal batch sizing

Diagnostic Procedures

Systematic diagnostic approaches enable efficient issue resolution and minimize service disruption during troubleshooting operations.

Diagnostic Steps:

1. Verify InferenceService status and event logs
2. Check resource allocation and utilization metrics
3. Validate network connectivity and endpoint accessibility
4. Review model loading logs and container status

How-To Verify InferenceServices and GPU usage

Implementation Patterns

Development Environments

Development and testing environments benefit from simplified configurations that prioritize rapid iteration over production-grade reliability and performance characteristics.

Development Configuration:

- Reduced resource allocation for cost efficiency
- Simplified networking without external access requirements
- Relaxed health check and timeout configurations
- Shared GPU resources across multiple development models

Production Deployments

Production environments require comprehensive configuration that ensures reliability, performance, and security while supporting operational requirements.

Production Requirements:

- High-availability configurations with appropriate redundancy
- Comprehensive monitoring and alerting integration
- Security controls aligned with organizational policies
- Capacity planning based on anticipated workload characteristics

Hybrid Architectures

Organizations may deploy models across multiple infrastructure tiers based on performance, security, or compliance requirements while maintaining consistent operational procedures.

Hybrid Considerations:

- Consistent model versions across different deployment environments
- Network connectivity and latency optimization between environments
- Security boundary management for cross-environment communication
- Unified monitoring and management across hybrid deployments

Getting Started

Initial Deployment

Organizations beginning with Model Serving should start with simple configurations and gradually add complexity based on operational experience and requirements.

Setup Sequence:

1. Verify Hybrid Manager and Asset Library installation
2. Configure basic ServingRuntime for target model framework
3. Deploy initial InferenceService with conservative resource allocation
4. Validate functionality through endpoint testing and monitoring

Configuration Resources

- *How-To Create an InferenceService for NVIDIA NIM Container*
- *How-To Setup GPU resources*
- *Accessing KServe endpoints (internal and external)*

Best Practices

- Start with proven ServingRuntime configurations before customization
- Implement comprehensive monitoring before production deployment
- Establish resource allocation guidelines based on model characteristics
- Document configuration decisions and operational procedures

Reference Documentation

Core Architecture:

- *Gen AI Architecture (Entities and Flows)*
- *Model Library Reference Manual*

Implementation Guides:

- *How-To Configure a ClusterServingRuntime*
- *How-To Create an InferenceService for NVIDIA NIM Container*
- *Observability for Model Serving*

Integration Documentation:

- *AI Accelerator Pipelines*
 - *Gen AI Builder*
 - *Hybrid Manager Asset Library*
-

Model Serving transforms governed AI models into production-ready inference capabilities that scale with organizational requirements while maintaining comprehensive control over performance, security, and operational characteristics.

Model Library Reference Manual

The **Model Library** serves as the central governance system for AI model images within EDB Postgres AI, providing organizational control over model lifecycles while enabling integration across Pipelines, Model Serving, Knowledge Bases, and Gen AI Builder.

Prerequisites: The Model Library requires *Hybrid Manager Asset Library* installation. All model meta-data, versioning, and lifecycle operations are managed through the Asset Library infrastructure.

Purpose and Function

Organizations implementing AI at scale face fundamental challenges with model management: fragmented repositories, version inconsistencies, compliance gaps, and limited operational visibility. The Model Library addresses these challenges through centralized governance that maintains control without sacrificing development velocity.

The system operates as a curated gateway between external model sources and internal AI capabilities, ensuring only validated models reach production environments while providing comprehensive audit trails for compliance requirements.

Core Architecture

Registry Integration Layer

The Model Library connects to external container registries through configurable synchronization rules. This layer automatically discovers, validates, and ingests model images based on organizational criteria while maintaining security boundaries.

Supported Registry Types:

- Harbor (private container registry)
- Amazon Elastic Container Registry (ECR)

- Azure Container Registry (ACR)
- Google Container Registry (GCR)
- Custom registry implementations

Governance Framework

Built-in approval workflows validate models against organizational policies before production availability. The framework supports complex multi-stakeholder approval processes while maintaining automated validation for standard compliance requirements.

Policy Enforcement Points:

- Security scanning and vulnerability assessment
- Performance benchmarking against organizational standards
- Compliance validation for regulatory requirements
- Digital signature verification for model provenance

Distribution Hub

Approved models become available across all EDB PG AI capabilities through standardized interfaces. This ensures consistent model versions across development, staging, and production environments while supporting environment-specific configurations.

Operational Workflows

Model Registration Process

Automated Discovery Repository rules define which external sources to monitor for new model versions. The system continuously scans configured registries and automatically ingests models matching defined criteria, reducing manual overhead while ensuring policy compliance.

Validation Pipeline Newly ingested models undergo automated validation including security scanning, metadata extraction, and performance verification. Models failing validation checks are quarantined until issues are resolved or explicitly approved through override procedures.

Approval Routing Validated models enter approval workflows based on classification and risk assessment. The system routes models to appropriate stakeholders while providing automated approval for models meeting predefined criteria.

Lifecycle Management

Version Control The system maintains immutable version history for all registered models, supporting semantic versioning schemes and backward compatibility validation. Version tracking enables rollback capabilities and dependency management across complex model ecosystems.

Promotion Pipelines Models advance through environment tiers (development → staging → production) based on validation checkpoints and approval gates. Automated promotion reduces deployment friction while maintaining quality control.

Deprecation Handling Configurable retention policies manage model lifecycle transitions, providing advance notification before deprecation and ensuring safe retirement without disrupting active deployments.

Integration Points

Model Serving Platform

The Model Library provides the curated catalog for *Model Serving Reference Manual* deployments. Only models approved through the governance framework can be deployed to production inference endpoints, ensuring consistent security and compliance standards.

Integration Benefits:

- Automated model updates with validation checkpoints
- Rollback capabilities for problematic deployments
- Performance metadata optimization for resource allocation
- Seamless KServe infrastructure integration

AI Accelerator Pipelines

Pipeline execution leverages validated models from the library, ensuring consistency across *AI Accelerator Pipelines* stages and environments. The system provides performance characteristics that optimize pipeline resource allocation and execution strategies.

Gen AI Builder

Gen AI Builder applications access governed model collections through the library interface. This ensures knowledge base implementations use validated embedding models and assistant creation references approved language models.

Security and Access Control

Multi-Tenant Architecture

Role-based access control aligns with organizational hierarchies while providing namespace isolation for different teams and projects. Fine-grained permissions control model registration, approval, and deployment activities based on user roles and organizational policies.

Security Validation

Automated vulnerability scanning examines all registered model images for known security issues. The system validates model signatures and provenance information during registration while implementing policy-based controls that prevent risky deployments.

Security Features:

- Digital signature verification for model authenticity
- Vulnerability database integration for threat detection
- Policy engine for automated security control enforcement
- Comprehensive audit logging for security monitoring

Implementation Considerations

Registry Configuration

Establishing registry connections requires careful consideration of network topology, authentication mechanisms, and synchronization frequency. Organizations must balance automation benefits with security requirements when configuring repository rules.

Critical Factors:

- Network connectivity and firewall requirements
- Authentication and authorization mechanisms
- Synchronization frequency and resource impact
- Error handling and retry strategies

How-To Integrate Private Registry with Model Library

Approval Workflow Design

Approval processes must balance governance requirements with development velocity. Organizations should design workflows that automate routine approvals while ensuring appropriate oversight for high-risk models.

Design Principles:

- Risk-based routing for efficient stakeholder involvement
- Automated approval for models meeting standard criteria
- Clear escalation paths for complex approval scenarios
- Comprehensive audit trails for compliance requirements

How-To Define Repository Rules for Model Library

Metadata Management

Comprehensive metadata capture supports discovery, governance, and operational requirements. Organizations should establish consistent tagging schemes and documentation standards that align with business requirements.

How-To Manage Repository and Image Tag Metadata

Operational Patterns**Enterprise Governance**

Large organizations benefit from the Model Library's sophisticated approval workflows and comprehensive audit capabilities. The system supports complex multi-stakeholder processes while maintaining development velocity through intelligent automation.

Typical Implementation:

1. Development teams register models in staging environments
2. Automated validation verifies security and compliance requirements
3. Risk-based routing directs models to appropriate approval stakeholders
4. Approved models propagate across all EDB PG AI capabilities
5. Production deployments automatically reference validated versions

Sovereign AI Deployments

Organizations requiring complete model sovereignty leverage private registry integration and air-gapped deployment capabilities. This pattern ensures sensitive models never leave organizational boundaries while maintaining enterprise operational capabilities.

Key Characteristics:

- Complete air-gap support with offline distribution
- Private registry integration with metadata preservation
- Automated compliance reporting for regulatory requirements
- Classified model handling procedures and controls

Development Consistency

Teams benefit from consistent model availability across development lifecycles. The Model Library eliminates version drift between environments while supporting environment-specific configurations and deployment strategies.

Limitations and Considerations

Performance Impact

Registry synchronization and model validation processes consume computational resources. Organizations must consider infrastructure capacity when configuring synchronization frequency and validation depth.

Network Dependencies

The system requires reliable network connectivity to external registries for synchronization operations. Network failures or connectivity issues can impact model availability and synchronization schedules.

Storage Requirements

Model image storage scales with organizational model inventory. Large organizations with extensive model catalogs should plan for significant storage requirements and associated backup strategies.

Getting Started

Initial Setup

1. Verify Hybrid Manager installation and configuration
2. Establish network connectivity to required external registries
3. Configure initial repository rules for model discovery
4. Implement basic approval workflows aligned with organizational requirements

Configuration Resources

- *[How-To Integrate Private Registry with Model Library](#)*
- *[How-To Define Repository Rules for Model Library](#)*
- *[How-To Manage Repository and Image Tag Metadata](#)*

Best Practices

- Implement least-privilege access controls for all model operations
- Establish regular security scanning schedules for registered models
- Maintain comprehensive backup and disaster recovery procedures
- Document approval workflows and escalation procedures clearly

Reference Documentation

Core Components:

- *Hybrid Manager Asset Library*
- *Gen AI Architecture (Entities and Flows)*

Integration Capabilities:

- *Model Serving Reference Manual*
- *AI Accelerator Pipelines*
- *Gen AI Builder*

Implementation Guides:

- *Model Library Reference Manual*
- *How-To Define Repository Rules for Model Library*

The Model Library transforms fragmented model management into systematic governance that scales with organizational AI initiatives while maintaining security, compliance, and operational excellence across all EDB Postgres AI capabilities.

Observability for Model Serving

Observability helps you ensure that your deployed AI models are running efficiently and reliably within AI Factory.

Model Serving in AI Factory uses KServe on Kubernetes to serve models. This provides built-in options to monitor:

- Model serving status and availability
- Resource usage (CPU, Memory, GPU)
- Inference performance and throughput

Key monitoring capabilities

KServe InferenceService status

You can inspect model serving status directly via Kubernetes:

```
kubectl get inferencesservice -n <namespace>
```

Common status fields include:

- Ready / NotReady
- URL endpoint
- Current replicas
- Allocated resources (GPU, CPU, Memory)

For detailed inspection:

```
kubectl describe inferencesservice <name> -n <namespace>
```

GPU utilization monitoring

If your models are deployed on GPU nodes, monitor GPU usage to optimize resource allocation.

Example:

```
kubectl top node
```

For deeper GPU-specific metrics (if supported):

```
nvidia-smi
```

Prometheus and Grafana integration

If Prometheus is configured, AI Factory model serving exposes metrics through KServe:

Prometheus annotations to enable scraping:

```
serving.kserve.io/enable-prometheus-scraping: "true"
prometheus.kserve.io/port: "8000"
prometheus.kserve.io/path: "/v1/metrics"
```

You can build Grafana dashboards to monitor:

- Inference requests per second
- Latency and error rates
- GPU utilization trends
- Pod restarts and health

Logs and debugging

You can access detailed logs from model serving pods:

```
kubectl logs -f <pod-name> -n <namespace>
```

Use logs to:

- Debug model loading or initialization issues
- Review inference request behavior
- Monitor response handling and error conditions

Hybrid Manager observability integration

When running AI Factory in conjunction with Hybrid Manager (HM), you can integrate model serving observability into the broader HM observability stack.

See [How-To Verify InferenceServices and GPU usage](#) for additional guidance on using HM observability tools with Model Serving.

Best practices

- Enable Prometheus scraping on InferenceServices for production.
- Regularly monitor GPU usage to optimize capacity.
- Set proper resource requests/limits in InferenceService specs.
- Leverage Grafana dashboards for real-time observability.
- Use log inspection to support debugging and tuning.

Related links

- [How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe](#)
- [Model Serving Concepts](#)
- [How-To Verify InferenceServices and GPU usage](#)

By applying these observability practices, you can operate your AI Factory Model Serving workloads with greater confidence, proactively identifying and addressing issues before they impact users.

Explainers

Model Serving Explained

Model Serving in **AI Factory** allows you to deploy AI models as scalable, production-grade inference services — running on your Kubernetes infrastructure.

It provides a **Kubernetes-native architecture** based on **KServe**, giving your models the ability to serve predictions and embeddings over network-accessible APIs.

AI Factory Model Serving is optimized to support enterprise-class AI workloads with:

- **GPU-accelerated infrastructure**
- **Flexible scaling**
- **Integrated observability**
- **Sovereign AI alignment** — models run under your governance
- **Seamless integration** with Gen AI Builder, Knowledge Bases, and other AI Factory pipelines

Before you start

Prerequisites for understanding Model Serving:

- Familiarity with **Kubernetes** basics
- Understanding of **KServe** and **InferenceService**
- Awareness of **Model Library** → Model Serving workflow in AI Factory
- Understanding of **Sovereign AI** principles — models running under your governance

Suggested reading:

- [Model Serving Concepts](#)

- *Model Library Explained In Depth*
- *Fundamental AI Concepts*

How Model Serving works

Core stack

Layer	Purpose
AI Factory	Provides infrastructure and Model Serving APIs
Hybrid Manager Kubernetes Cluster	Hosts model-serving workloads
KServe	Manages model serving lifecycle and APIs
InferenceService	Deployed model resource
Model Library	Manages model image versions
GPU Nodes	Run high-performance model serving pods
User Applications	Call model endpoints via REST/gRPC

Key components

- **InferenceService** — Kubernetes CRD representing a deployed model.
- **ServingRuntime / ClusterServingRuntime** — Define reusable runtime configurations.
- **Model containers** — Currently focused on **NVIDIA NIM** containers in AI Factory 1.2.
- **Observability** — Integrated Prometheus-compatible metrics, Kubernetes logging.

Supported models

AI Factory Model Serving currently supports **NVIDIA NIM** containers for:

Model Type	Example Usage
Text Completion	LLM agents, Assistants
Text Embeddings	Knowledge Bases, RAG
Text Reranking	RAG pipelines
Image Embeddings	Multi-modal search
Image OCR	Document extraction

See: [Supported models](#)

Deployment architecture

Applications → Model Endpoints (REST/gRPC) → KServe → GPU-enabled Kubernetes → Model Containers

- Each model is isolated in its own **InferenceService**.
- KServe manages:

- Model lifecycle (start, stop, update)
- Scaling (including scale-to-zero)
- Endpoint routing (REST/gRPC)
- GPU resources are provisioned and scheduled via Hybrid Manager integration.

Patterns of use

Gen AI Builder

- LLM endpoints power **Assistants** and **Agents**.
- Embedding models support hybrid RAG pipelines.

Knowledge Bases

- Embedding models serve **vectorization** needs.
- Retrieval and reranking models power **semantic search** pipelines.

Custom applications

- Business applications can consume **InferenceService** endpoints for:
- Real-time predictions
- Image analysis
- Text processing

Best practices

- Deploy models via the **Model Library** → **Model Serving** flow to ensure governance.
- Use **ClusterServingRuntime** for reusable runtime configs.
- Monitor **GPU utilization** and model latency closely.
- Test **scale-to-zero** configurations for readiness in production.
- Ensure **Model Library tags** are versioned and documented.
- Regularly audit deployed **InferenceServices** as part of Sovereign AI governance.

Summary

Model Serving in AI Factory provides a robust, scalable architecture for serving **production AI models**:

- Kubernetes-native serving with KServe
- GPU acceleration and optimized serving runtimes
- Integrated observability and governance
- Tight integration with AI Factory components: Gen AI Builder, Knowledge Bases, custom AI pipelines

Model Serving helps you implement **Sovereign AI** — with your models, on your infrastructure, under your control.

Next steps

- *How-To Create an InferenceService for NVIDIA NIM Container*
- *How-To Verify InferenceServices and GPU usage*
- *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe*
- *Model Library Explained In Depth*

Model Serving gives you a powerful foundation for building intelligent applications and data products — securely, scalably, and under your governance — as part of **EDB Postgres® AI**.

Model Library Explained In Depth

The **Model Library** in **AI Factory** is your central interface for discovering, managing, and deploying AI models within EDB PG AI.

It is powered by the core **Image and Model Library** of Hybrid Manager (HM). The Model Library presents a curated, AI-focused view of model images validated for production use.

Model images deployed through the Model Library power AI Factory features including:

- **Gen AI Builder** Assistants and pipelines
- **Knowledge Base pipelines** (embeddings, RAG)
- **Other AI Factory applications** using Model Serving

Before you start

Prerequisites for understanding Model Library:

- Basic familiarity with **Model Serving** and KServe concepts
- Understanding of **InferenceService** and runtime models
- Awareness of **Image and Model Library** architecture in Hybrid Manager
- Helpful: Awareness of **Sovereign AI principles**

Suggested path:

- *Fundamental AI Concepts*
- *Model Serving Concepts*

What is the Model Library?

The Model Library:

- Presents a **curated AI view** of the broader Image and Model Library

- Allows users to deploy validated model images to the **Model Serving (KServe)** layer
- Provides lifecycle management for models powering AI Factory experiences

Key point: Model Library is not a separate registry. It is a controlled *view* on top of the **governed Image and Model Library**.

All model images:

- Flow through the core Asset Library path
- Are subject to platform-wide **governance and security policies**
- Become available in Model Library only when validated for AI Factory use

How does it work?

Architecture flow:

```
Container Registry → Image and Model Library → Model Library → Model Serving (KServe) → AI Factory Workloads
```

Layer	Role
Container Registry	Stores model container images (e.g., NVIDIA NIM, open models)
Image and Model Library	Single source of truth for all container images
Model Library	AI-focused console for validated model images
Model Serving (KServe)	Runs deployed model instances as InferenceServices
AI Factory Workloads	Gen AI Builder, Knowledge Base pipelines, Assistants, AI applications

Workflow:

1. Images are ingested into **Asset Library** (private or public registries).
2. Images validated for AI model serving appear in **Model Library**.
3. Users browse, select, and deploy models to **Model Serving** via KServe.
4. Deployed models power:
 - **Knowledge Base ingestion and retrieval**
 - **Gen AI Builder Assistants and pipelines**
 - **Other AI Factory capabilities**

Why use the Model Library?

- **Unified governance** — All images flow through the same governance process.
- **Security and auditability** — Registry integration, tag selection, and image audit are unified.
- **Sovereign AI** — You control which models are used, how they are deployed, and where they run.
- **Consistency** — Database and AI models share the same container image management system.
- **Flexibility** — You can add your own model images via Asset Library path and control their availability in AI Factory.

What does the Model Library provide?

Users can:

- Browse available **AI model images**
- View supported **tags and versions**
- Deploy models to **Model Serving** infrastructure
- Manage model deployment lifecycle
- Understand **which models power current AI Factory workloads**, including:
 - **Knowledge Bases** (via AIDB ingestion and retrieval)
 - **Gen AI Builder** (for Assistants and pipelines)

Supported model types

Model Type	Example Image
Text Completion	llama-3.3-nemotron-super-49b
Text Embedding	arctic-embed-l
Image Embedding	nvclip
OCR	paddleocr
Text Reranker	llama-3.2-nv-rerankqa-1b-v2

- Support for additional model types is planned (vision, multi-modal, RAG-optimized).
- **Custom models** can be added via private registry integration.

Patterns of use

Knowledge Bases

- Embedding and retrieval models used in **Knowledge Base indexing** and **RAG** pipelines.
- Example: Use **arctic-embed-l** or **nvclip** models.

Gen AI Builder Assistants

- Power Assistants with models deployed through Model Library:
- LLMs for text generation (e.g., llama-3)
- OCR or Vision models (e.g., paddleocr)
- Custom tools using Model Serving endpoints

Hybrid + Sovereign AI alignment

- All models run on **your infrastructure** via Hybrid Manager KServe layer.
- You control:
 - Which models are deployed
 - Resource allocations (CPU/GPU)
 - Deployment topology
 - Observability and auditing

Custom model integration

- Use **Integrate Private Registry** flow to add models from your own registry.
- Define **Repository Rules** to control image availability.
- Ensure images pass governance checks before appearing in Model Library.

Best practices

- Maintain **clear governance** over model images.
- Use **Repository Rules** to manage visibility and compliance.
- Monitor **model usage** and resource consumption in Hybrid Manager dashboards.
- Version and document models used in production.
- Test custom models thoroughly before enabling for AI Factory workloads.

Model Serving integration

Models deployed from Model Library run on the **Model Serving (KServe)** layer:

- Guided workflow provided in Model Library console:
 1. Select model image and tag.
 2. Configure runtime (replicas, resources, GPUs, etc.).
 3. Deploy → creates **InferenceService** in HM Kubernetes environment.
- Deployed models power:
 - **AIDB Knowledge Base ingestion**
 - **Gen AI Builder Assistants and pipelines**
 - **Other AI Factory services**

Related topics

- *Model Library Reference Manual*
- *How-To Deploy AI Models from Model Library*
- *How-To Integrate Private Registry with Model Library*
- *How-To Define Repository Rules for Model Library*
- *Model Serving Concepts*
- *Pipelines models reference*

Next steps

- Start by browsing available models in your **Model Library**.
- Deploy your first model for **Knowledge Base** ingestion or **Gen AI Builder Assistants**.
- Explore **Model Serving How-To Guides** to optimize deployment.
- Define governance and compliance policies for model image flow.

How to set up serving

How-To Integrate Private Registry with Model Library

Integrate Private Registry with Model Library

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

This guide explains how to integrate your organization's **private container registry** with the AI Factory **Model Library**.

Once integrated, your custom-built or internally-approved model images will appear in the Model Library console, ready to deploy into Model Serving.

Who should use this guide?

- AI platform admins responsible for container registry governance.
- DevOps engineers managing private registries for AI images.
- Developers needing to deploy private AI models through AI Factory.

What this enables

- You can register private registries with Hybrid Manager (HM).
- HM can discover AI model images in your registry.
- These images will be available in the **Model Library** for deployment to Model Serving.
- You can control visibility and usage of private model images across your AI Factory workloads.

Estimated time to complete

10–15 minutes per registry configuration.

Prerequisites

Before you begin:

- You must have admin access to your private container registry (AWS ECR, GCP GCR, Azure ACR, Harbor, or similar).
- You must have admin access to HM (to configure registry integration).
- You must know the required credentials (username/password or token) for your registry.
- Your registry must expose a compatible Docker Registry API v2 endpoint.

Supported registry types

- AWS Elastic Container Registry (ECR)
- Google Container Registry (GCR) / Artifact Registry
- Azure Container Registry (ACR)
- Harbor
- Generic registries supporting Docker Registry API v2

Integration Steps

1. Prepare registry credentials

- For public registries → no authentication required.
- For private registries → prepare one of:
 - Username/password pair
 - Personal access token
 - Robot account credentials (Harbor)

Confirm you can perform a `docker pull` of your model images locally using these credentials.

2. Register your registry in HM

- In the console, go to:

AI Factory > Model Library > Manage Repositories > Add Registry

You will see a dialog prompting for:

Field	Description
Registry URL	The full registry hostname (e.g., <i>myregistry.company.com</i>)
Registry Type	Select from supported registry types
Username	(Optional) Username for authentication
Password/Token	(Optional) Password or token for authentication
Registry Name (Label)	Friendly name displayed in Model Library console

- Fill in the required fields.

- Click **Add Registry**.

3. Verify registry integration

After adding:

- HM will attempt to connect to the registry and validate credentials.
- If successful, your registry will appear in the **Manage Repositories** list.
- HM will begin periodic sync of repository tags from this registry.

4. Define Repository Rules

To control which repositories/tags are discovered:

- After adding the registry, configure **Repository Rules**.
- See: *How-To Define Repository Rules for Model Library*

5. Validate image availability

After sync completes:

- Go to **AI Factory > Model Library**.
- Select the Registry scope or Repository filter.
- Confirm your private model images appear with correct tags.

You can now deploy these images via the normal **Deploy to Model Serving** flow.

Tips & Best Practices

- Use **robot accounts** or **token-based auth** when possible to avoid exposing personal credentials.
- Limit discovery scope via **Repository Rules** — avoid syncing the entire registry.
- Tag images clearly with version info to aid selection in Model Library.
- For multi-tenant environments, segment registry visibility carefully.

Troubleshooting

Registry connection failed

- Check Registry URL (must not include `https://`, just the hostname).
- Validate credentials by testing `docker login` manually.
- Ensure Registry API v2 is enabled.

Images not appearing

- Check that Repository Rules allow the relevant repository.
- Verify image has a valid tag.
- Confirm periodic sync has completed.

Authentication errors

- For AWS ECR → ensure IAM permissions allow `ecr:GetAuthorizationToken`.
- For Azure ACR → ensure token has `acrPull` role.
- For GCR → use a service account with Artifact Registry access.

Summary

- You can integrate private registries with Model Library.
- Your private model images become available to deploy via Model Serving.
- You can govern visibility via Repository Rules.
- Model Library unifies both public and private image sources for your AI workloads.

Related Links

- [*Model Library Explained In Depth*](#)
- [*How-To Define Repository Rules for Model Library*](#)
- [*Model Library Explained In Depth*](#)
- [*How-To Deploy AI Models from Model Library*](#)

How-To Define Repository Rules for Model Library

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See [/edb-postgres-ai/1.3/hybrid-manager/ai-factory/](#).

This guide explains how to define **Repository Rules** to control which model image repositories and tags from your connected registries appear in the AI Factory **Model Library**.

Repository Rules help you manage:

- **Scope** of image discovery
- **Governance** over which images are exposed
- **Version control** by controlling which tags are visible

Who should use this guide?

- AI platform admins integrating private registries
- DevOps engineers managing container registry integrations
- Security/compliance teams governing AI model image usage

What this enables

- You can precisely control which image repositories HM syncs.
- You can filter which tags within each repository are exposed.
- You can govern which images are available in the Model Library for deployment.

Estimated time to complete

5–10 minutes per Repository Rule.

Prerequisites

- You must have already *How-To Integrate Private Registry with Model Library*.
- Your registry must appear in **AI Factory > Model Library > Manage Repositories**.

Understanding Repository Rules

Repository Rules are attached to each configured registry.

They allow you to define:

- **Which repositories** to discover
- **Which tags** within those repositories to expose

This prevents accidental exposure of test images, unvetted model builds, or entire registry contents.

Repository Rules are **evaluated periodically** — newly pushed images matching the rule will appear in Model Library on the next sync.

How Repository Rules work

Each rule typically includes:

Field	Description
Repository Name	Name of the repository to include (exact match or pattern)
Tag Filter	(Optional) Filter for tag names (exact match, prefix, regex)
Rule Enabled	Whether this rule is currently active

Example:

Repository Name	Tag Filter
<code>my-company/llama3-models</code>	<code>^release-.*</code>

→ Only tags starting with `release-` in `my-company/llama3-models` will appear.

Defining Repository Rules

1. Navigate to Repository Rules

- Go to:

AI Factory > Model Library > Manage Repositories

- Click **Manage Rules** for your desired registry.

2. Add a Repository Rule

- Click **Add Rule**.
- Fill in:
 - **Repository Name** — required
 - **Tag Filter** — optional
- Enable/Disable toggle
- Click **Save**.

The rule will appear in the rules list.

3. Wait for next sync or trigger manual sync

HM will apply rules on the next registry sync cycle.

If your UI supports manual sync:

- Click **Sync Now** after adding/updating a rule.

4. Verify Model Library content

- Go to **AI Factory > Model Library**.
- Use filters to browse images from your registry.
- Confirm only desired repositories/tags appear.

Example Use Cases

Basic: Restrict to a single repo

Repository Name	Tag Filter
<i>my-org/nim-models</i>	(blank — all tags)

→ All tags in `my-org/nim-models` will appear.

Advanced: Filter to production tags

Repository Name	Tag Filter
<i>models/gpt4</i>	<i>^prod-.*</i>

→ Only tags starting with `prod-` will appear.

Multi-repo governance

- Add multiple Repository Rules per registry as needed.
- Use Tag Filters to align with your model promotion pipelines (dev, staging, prod).

Tips & Best Practices

- **Be explicit** — add Repository Rules for only the repositories you want to expose.
- Use **Tag Filters** to enforce version governance and CI/CD promotion stages.
- Regularly **audit Repository Rules** to ensure compliance with internal image standards.
- If your registry uses structured tagging (e.g., `prod-`, `test-`), leverage Tag Filters to separate experimental from production models.

Troubleshooting

Images not appearing

- Check that Repository Rule exists for the desired repo.
- If using Tag Filter — verify tags match the filter exactly.
- Check when the last sync ran; trigger manual sync if needed.

Too many images appearing

- Refine Repository Rules.
- Add Tag Filters to narrow scope.

Repository Rule has no effect

- Confirm repository name is exact (case sensitive).
- Ensure Rule is enabled.
- Trigger manual sync and check HM logs if issue persists.

Summary

- Repository Rules govern which image repositories and tags appear in Model Library.
- You can scope exposure by repository name and tag filters.
- This supports governance, version control, and security best practices.
- Fine-grained control helps ensure only vetted model images are deployed.

Related Links

- *[Model Library Explained In Depth](#)*
- *[How-To Integrate Private Registry with Model Library](#)*
- *[How-To Deploy AI Models from Model Library](#)*
- *[Model Library Explained In Depth](#)*

How-To Manage Repository and Image Tag Metadata

Manage Repository and Image Tag Metadata

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

This guide explains how to manage metadata for your repositories and image tags in the Image and Model Library. Proper metadata helps you track, organize, and govern images used for AI model serving and Postgres cluster provisioning.

Who should use this guide?

- Platform admins curating approved image sets.
- DevOps teams tagging images for specific environments (Dev, Test, Prod).
- AI teams managing model lifecycle in AI Factory.
- Security & compliance owners applying ownership metadata.

What this enables

- Apply descriptive tags to repositories and image tags.
- Edit READMEs for repositories for team guidance.
- Track image ownership and promotion state.
- Provide clear context to end users in Image Library console.

Estimated time to complete

5–10 minutes per repository/image.

Prerequisites

- You have access to Image and Model Library in Hybrid Manager or AI Factory.
- Your repository integration is already configured and image sync completed.
- You have appropriate permissions to edit metadata.

Supported Metadata Features

Metadata Type	Applies To
HM Tags	Repository, Image Tag
README	Repository

Managing Repository Metadata

Apply HM Tags to a Repository

1. Navigate to Image and Model Library.
2. Click on desired Repository (e.g. `postgresql,meta/llama-3.3,your-private-model-repo`).
3. Click **Manage Tags** or **Edit Tags**.
4. Add HM Tags:
 - Example: `approved-production,beta,team-ai,archived`.
5. Save changes.

Edit Repository README

1. On the Repository page, click **Edit README**.
2. Enter or update README content (supports Markdown).
 - Example sections:
 - Purpose of repository.
 - Model or DB image types contained.
 - Support contacts.
 - Security notes.
3. Save README. It will now display on Repository details page.

Managing Image Tag Metadata

Apply HM Tags to an Image Tag

1. Navigate to Repository.
2. Click **Image Tags** tab.
3. For desired Image Tag, click **Manage Tags** or **Edit Tags**.
4. Add HM Tags:
 - Example: `approved-production`, `testing`, `deprecated`.
5. Save changes.

Common Tag Usage Patterns

Tag Example	Usage
<i>approved-production</i>	Images approved for Production use.
<i>beta</i>	Images under evaluation.
<i>team-ai</i>	Models used by AI team.
<i>archived</i>	Images no longer in active use.
<i>test-only</i>	Images used for testing / PoC deployments.

Best Practices

- Always apply `approved-production` only after full validation.
- Clearly tag deprecated or archived images.
- Use team-based tags (`team-ml`, `team-dba`) to indicate ownership.
- Add README to every custom Repository — improves transparency.
- Use Image Tags + HM Tags for precise version tracking (ex: `postgresql:17.4-xxxx` tagged with `approved-production`).

Limitations & Notes

- Applying Tags / README does not affect image pull or runtime behavior — they are for UI and governance.
- Tags must follow naming conventions established in your org.
- Currently HM Tags do not propagate to private registry — they are stored in HM metadata.

Example Workflow

Scenario: AI team is preparing to move a new Reranker model into production.

1. Team syncs model image `nv-rerankqa-1b-v2:1.8.2` into `private-model-repo`.
2. After test validation:
 - Apply `approved-production` tag to this Image Tag.
 - Apply `team-ai` to Repository and Image Tag.
 - Edit Repository README to include model version history and ownership.
3. Now AI Factory users can easily identify `approved-production` Reranker model.

Summary

- You can manage HM Tags and READMEs for Repositories and Image Tags.
- This improves governance, tracking, and transparency.
- Always tag Production-approved images clearly.
- Keep READMEs updated for team guidance.

Related Links

- [How-To Deploy AI Models from Model Library](#)
- [How-To Integrate Private Registry with Model Library](#)
- [How-To Define Repository Rules for Model Library](#)
- [Model Library Explained In Depth](#)
- [Model Library Explained In Depth](#)

How-To Setup GPU resources

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Use this guide to prepare GPU resources in your Kubernetes cluster (Hybrid Manager or compatible) to support Model Serving with KServe.

Goal

Prepare your cluster to run GPU-based Model Serving workloads using KServe.

Estimated time

20–40 minutes (provisioning depends on your cloud provider).

What you accomplish

- Provision GPU node groups/pools in your cluster.
- Label and taint GPU nodes correctly.
- Deploy the NVIDIA device plugin DaemonSet.
- Store your NVIDIA API key as a Kubernetes secret.
- Enable your cluster to run NIM model containers in KServe.

Prerequisites

- Access to a Kubernetes cluster with appropriate permissions.
- Administrative access to provision node groups (AWS EKS / GCP GKE / RHOS).
- NVIDIA API key for accessing NIM models.
- Familiarity with `kubectl`.

Provision GPU nodes

Provision GPU node groups (EKS) or node pools (GKE/RHOS):

- Use instances with L40S or A100 GPUs (for example, `g6e.12xlarge` on AWS or `a2-highgpu-4g` on GCP).
- Recommended: at least one node with four GPUs for large models.

Label and taint GPU nodes

```
kubectl label node <gpu-node-name> nvidia.com/gpu=true
kubectl taint nodes <gpu-node-name> nvidia.com/gpu=true:NoSchedule
```

Deploy the NVIDIA device plugin

```
kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.14.1/
↪nvidia-device-plugin.yml
kubectl get ds -n kube-system nvidia-device-plugin-daemonset
```

Store NVIDIA API key as Kubernetes secret

```
kubectl create secret generic nvidia-nim-secrets --from-literal=NGC_API_KEY=<your_
↪NVIDIA_API_KEY>
```

This secret is used by `ClusterServingRuntime` for NIM models.

Related concepts

- [Model Serving Reference Manual](#)

Next steps

- [How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.comcontainer with KServe](#)
- [How-To Verify InferenceServices and GPU usage](#)

How-To Configure a ClusterServingRuntime

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See [/edb-postgres-ai/1.3/hybrid-manager/ai-factory/](#).

This guide explains how to configure a **ClusterServingRuntime** in KServe. A ClusterServingRuntime defines the environment used to serve your AI models — specifying container image, resource settings, environment variables, and supported model formats.

For Hybrid Manager users, configuring runtimes is a core step toward enabling Model Serving — see [Pipelines models reference](#).

Goal

Configure a ClusterServingRuntime so it can be used by InferenceServices to deploy models.

Estimated time

5–10 minutes.

What you will accomplish

- Define a ClusterServingRuntime YAML manifest.
- Apply it to your Kubernetes cluster.
- Enable reusable serving configuration for one or more models.

What this unlocks

- Supports consistent deployment of models using a standard runtime definition.
- Allows for centralized control over serving images and resource profiles.
- Required step for deploying NVIDIA NIM containers with KServe.

Prerequisites

- Kubernetes cluster with KServe installed.
- Access to container image registry with the desired model server image.
- NVIDIA GPU node pool configured (if using GPU-based models).
- (If required) Kubernetes secret configured for API keys (e.g., [build.nvidia.com](#)).

For background concepts, see:

- [Model Serving Concepts](#)
- [AI Factory terminology](#)

Steps

1. Create ClusterServingRuntime YAML

Create a file named `ClusterServingRuntime.yaml`.

Example:

```
apiVersion: serving.kserve.io/v1alpha1
kind: ClusterServingRuntime
metadata:
  name: nvidia-nim-llama-3.1-8b-instruct-1.3.3
  namespace: default
spec:
  containers:
    - env:
        - name: NIM_CACHE_PATH
          value: /tmp
        - name: NGC_API_KEY
          valueFrom:
            secretKeyRef:
              name: nvidia-nim-secrets
              key: NGC_API_KEY
      image: upmdev.azurecr.io/nim/meta/llama-3.1-8b-instruct:1.3.3
      name: kserve-container
      ports:
        - containerPort: 8000
          protocol: TCP
      resources:
        limits:
          cpu: "12"
          memory: 64Gi
        requests:
          cpu: "12"
          memory: 64Gi
        volumeMounts:
          - mountPath: /dev/shm
            name: dshm
  imagePullSecrets:
    - name: edb-cred
  protocolVersions:
    - v2
    - grpc-v2
  supportedModelFormats:
    - autoSelect: true
      name: nvidia-nim-llama-3.1-8b-instruct
      priority: 1
      version: "1.3.3"
  volumes:
    - emptyDir:
        medium: Memory
        sizeLimit: 16Gi
      name: dshm
```

Key fields explained:

- `containers.image` : The model server container (e.g., NVIDIA NIM image).
- `resources` : CPU, memory, and GPU requirements.
- `NGC_API_KEY` : Secret reference for NVIDIA models.
- `supportedModelFormats` : Logical name used by `InferenceService` to reference this runtime.

2. Apply the ClusterServingRuntime

Run:

```
kubectl apply -f ClusterServingRuntime.yaml
```

3. Verify deployed ClusterServingRuntime

Run:

```
kubectl get ClusterServingRuntime
__OUTPUT__
NAME                                     AGE
nvidia-nim-llama-3.1-8b-instruct-1.3.3  1m
```

You can inspect full details with:

```
kubectl get ClusterServingRuntime <name> -o yaml
```

4. Reference runtime in InferenceService

When you create your `InferenceService`, reference this runtime:

```
runtime: nvidia-nim-llama-3.1-8b-instruct-1.3.3
modelFormat:
name: nvidia-nim-llama-3.1-8b-instruct
```

See *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.comcontainer with KServe* .

Notes

- Runtimes are reusable — you can deploy multiple models referencing the same `ClusterServingRuntime`.
- Use meaningful names and version fields in `supportedModelFormats` for traceability.
- You can update a runtime by editing and re-applying the YAML.

Next steps

- *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.comcontainer with KServe*
- *How-To Update GPU Resources for an InferenceService*
- *How-To Monitor deployed models with KServe*

Related reading

- [Model Serving Concepts](#)
- [AI Factory terminology](#)

How to deploy model

How-To Deploy AI Models from Model Library

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

This guide explains how to deploy AI models from the AI Factory **Model Library** into Model Serving (powered by KServe) in your Hybrid Manager (HM) environment.

Once deployed, these models power key AI Factory features:

- **Knowledge Bases** (via AIDB pipelines)
- **Gen AI Builder Assistants and pipelines**
- Other AI Factory and application integrations

Who should use this guide?

- AI platform admins deploying validated model images
- Data engineers configuring AI models for Knowledge Bases
- AI application developers configuring models for Assistants

What this enables

Once deployed:

- Your AI models are available in Model Serving.
- You can link them to Knowledge Bases or Gen AI Builder pipelines.
- You can monitor and manage deployed models via the HM Model Serving UI or Kubernetes.

Estimated time to complete

10–20 minutes per model, depending on model size and cluster resources.

Prerequisites

Before you begin:

- An active HM environment with **GPU worker nodes** configured.
- build.nvidia.com API key stored as `nvidia-nim-secrets` in Kubernetes.
- The model image you want to deploy must be visible in the **Model Library**.
- KServe must be configured and ready in your cluster.

→ For a full setup guide, see: [How-To Setup GPU resources](#)

Steps to deploy an AI model

1. Browse and select model in Model Library

- Go to **AI Factory > Model Library**.
- Browse available model images.
- Review versions and tags.
- Select the version you want to deploy.

2. Configure and deploy model

- Click **Deploy** or **Deploy to Model Serving**.
- Configure deployment parameters:
 - Number of replicas
 - Resource requests/limits (GPU, CPU, memory)
 - Model runtime settings (if needed)
- Confirm and deploy.

This triggers creation of:

- A `ClusterServingRuntime` for the model (if not already defined).
- An `InferenceService` for the specific deployment.

3. Verify deployed model

You can verify your deployed models using:

Model Serving UI in HM

- Go to **AI Factory > Model Serving** (or Hybrid Manager > AI Factory > Model Serving).
- Confirm model appears with status **Ready**.

Or use kubectl:

```
kubectl get InferenceService -n default
__OUTPUT__
NAME STATUS URL GPUs
meta-nim-llama-3-3-nemotron-super-49b Ready http://meta-nim-llama-3-3-nemotron-super-
→49b-predictor... 4
nim-snowflake-arctic-embed-1 Ready http://nim-snowflake-arctic-embed-1-predictor... 1
...
```

4. Connect model to AI Factory workloads

Once the model is Ready:

- You can select it in:
 - Knowledge Base pipelines (for embedding or reranking)
 - Gen AI Builder pipelines
 - Assistant configurations

→ The UI will show models available for each use case based on their **type** (Embedding, Completion, Reranking, etc.).

Supported model types

Model Type	Example Image
Text Completion	llama-3.3-nemotron-super-49b
Text Embedding	arctic-embed-l
Image Embedding	nvclip
OCR	paddleocr
Text Reranker	llama-3.2-nv-rerankqa-1b-v2

Tips & Best Practices

- **GPU placement:** Ensure your model matches your GPU capacity. Large models like **llama-3.3-49b** require multiple GPUs on a single node.
- **Quota management:** Limit number of large models deployed simultaneously to avoid overloading GPU nodes.
- **Version testing:** Test new model versions in isolated deployments before promoting to production pipelines or Assistants.

Troubleshooting

Model stuck in Pending

- Check GPU node taints/labels.
- Verify InferenceService tolerations and nodeSelectors match.

Model not appearing in Model Library

- Confirm image is correctly tagged and synced via Image and Model Library.
- Verify repository rules if using private registry.

Kubernetes errors on deploy

- Check `kubectl describe InferenceService <model>` for detailed error logs.

Summary

- You can deploy AI models from the AI Factory Model Library.
- Deployed models run via KServe Model Serving.
- Deployed models power Knowledge Bases and Gen AI Builder Assistants.
- The deployment flow ensures consistent governance and visibility.

Related Links

- [Model Library Explained In Depth](#)
- [Model Library Explained In Depth](#)
- [How-To Setup GPU resources](#)

- *How-To Deploy an NVIDIA NIM Microservices available on [build.nvidia.com](https://build.nvidia.com/container) container with KServe*

How-To Create an InferenceService for NVIDIA NIM Container

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

This How-To explains how to deploy an NVIDIA NIM container as an InferenceService using KServe on Kubernetes. You will define and apply an `InferenceService.yaml` manifest that references an existing ClusterServingRuntime and runs a GPU-accelerated model instance.

This is a core step to making your model available for inference through a network-accessible endpoint.

Goal

Deploy an InferenceService for an NVIDIA NIM container using KServe.

Estimated time

10–15 minutes.

What you will accomplish

- Define an InferenceService manifest.
- Deploy the InferenceService to your Kubernetes cluster.
- Make the model available via a network endpoint.

What this unlocks

Once deployed, your model can be consumed by applications (such as Gen AI Builder agents) via standard inference APIs. It will also appear in your KServe monitoring and can be scaled or updated dynamically.

Prerequisites

- Kubernetes cluster with KServe installed and configured.
- ClusterServingRuntime configured for your NIM container. See *How-To Configure a ClusterServingRuntime*.
- GPU-enabled node pool provisioned and labeled appropriately.
- `build.nvidia.com` API key stored as Kubernetes secret, if required.
- The desired NIM container image available in your registry.

Steps

1. Create InferenceService manifest

Create a file named `InferenceService.yaml` with the following content:

```
apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  annotations:
    serving.kserve.io/enable-prometheus-scraping: "true"
```

(continues on next page)

(continued from previous page)

```

    prometheus.kserve.io/port: "8000"
    prometheus.kserve.io/path: "/v1/metrics"
name: llama-3-1-8b-instruct-1xgpu-g5
namespace: default
spec:
  predictor:
    tolerations:
      - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
  nodeSelector:
    nvidia.com/gpu: "true"
  imagePullSecrets:
    - name: edb-cred
  model:
    modelFormat:
      name: nvidia-nim-llama-3.1-8b-instruct
    resources:
      limits:
        nvidia.com/gpu: "1"
      requests:
        nvidia.com/gpu: "1"
    runtime: nvidia-nim-llama-3.1-8b-instruct-1.3.3

```

Key points:

- **name** → provide a unique name for your InferenceService.
- **tolerations** and **nodeSelector** → must match your GPU node pool configuration.
- **runtime** and **modelFormat** → must match the values used in your ClusterServingRuntime.
- **GPU resources** → specify the number of GPUs to allocate.

2. Apply the InferenceService

Run:

```
kubectl apply -f InferenceService.yaml
```

3. Verify the deployment

Check the InferenceService status:

```

kubectl get InferenceService -o custom-columns=NAME:.metadata.name,MODEL:.spec.
↪predictor.model.modelFormat.name,URL:.status.address.url,RUNTIME:.spec.predictor.
↪model.runtime,GPU:.spec.predictor.model.resources.limits.nvidia\.com/gpu --
↪namespace=default

```

You should see:

- The name of your InferenceService.
- The model format being served.
- The endpoint URL.

- The ClusterServingRuntime used.
- The number of GPUs allocated.

What's next

Now that your model is deployed:

- Your application (or Gen AI Builder agent) can send inference requests to the provided URL.
- You can monitor model performance and resource usage.
- You can update the GPU allocation or other runtime parameters if needed.

Related topics

- *How-To Configure a ClusterServingRuntime*
- *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.comcontainer with KServe*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *AI Factory terminology*

Next steps

- *How-To Update GPU Resources for an InferenceService*
- *How-To Monitor deployed models with KServe*
- *How-To Create an InferenceService for NVIDIA NIM Container*

Explore more in the [AI Factory learning guide](#) .

How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.comcontainer with KServe

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

This guide explains how to deploy an NVIDIA NIM Microservices available on build.nvidia.com container using KServe on a Kubernetes cluster.

While the steps here apply to general Kubernetes environments, in Hybrid Manager AI Factory, we provide additional value such as lifecycle management, observability, and simplified integration. Learn more in the [Model Serving in Hybrid Manager](#) section.

Goal

Deploy an NVIDIA NIM container using KServe to create a network-accessible inference service that can be consumed by applications.

Estimated time

15–30 minutes depending on cluster setup.

What you will accomplish

- Define and deploy a ClusterServingRuntime for an NVIDIA NIM container.
- Deploy an InferenceService that uses this runtime.
- Validate your deployment and retrieve the model endpoint.

What this unlocks

- Ability to serve NVIDIA NIM via standard inference protocols (HTTP/gRPC).
- Prepare to integrate these models with applications or tools such as Griptape (Gen AI Builder) or AIDB Knowledge Bases.
- Foundation for using Hybrid Manager AI Factory model-serving capabilities.

Prerequisites

- Kubernetes cluster with KServe installed.
- GPU node pool configured (with NVIDIA device plugin).
- NVIDIA NIM container image available in a private registry or NGC.
- Kubernetes secret containing NGC API Key.
- kubectl configured for your cluster.

For background concepts, see:

- [Model Serving Concepts](#)
- [AI Factory terminology](#)

Steps

1. Create ClusterServingRuntime

Define ClusterServingRuntime.yaml:

```
apiVersion: serving.kserve.io/v1alpha1
kind: ClusterServingRuntime
metadata:
  name: nvidia-nim-llama-3.1-8b-instruct-1.3.3
  namespace: default
spec:
  containers:
    - env:
      - name: NIM_CACHE_PATH
        value: /tmp
      - name: NGC_API_KEY
        valueFrom:
          secretKeyRef:
            name: nvidia-nim-secrets
            key: NGC_API_KEY
      image: your-registry/nim/meta/llama-3.1-8b-instruct:1.3.3
      name: kserve-container
      ports:
```

(continues on next page)

(continued from previous page)

```

    - containerPort: 8000
      protocol: TCP
  resources:
    limits:
      cpu: "12"
      memory: 64Gi
    requests:
      cpu: "12"
      memory: 64Gi
  volumeMounts:
    - mountPath: /dev/shm
      name: dshm
imagePullSecrets:
  - name: edb-cred
protocolVersions:
  - v2
  - grpc-v2
supportedModelFormats:
  - autoSelect: true
    name: nvidia-nim-llama-3.1-8b-instruct
    priority: 1
    version: "1.3.3"
volumes:
  - emptyDir:
      medium: Memory
      sizeLimit: 16Gi
    name: dshm

```

Apply the runtime:

```
kubect1 apply -f ClusterServingRuntime.yaml
```

2. Create InferenceService

Define InferenceService.yaml :

```

apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  annotations:
    serving.kserve.io/enable-prometheus-scraping: "true"
    prometheus.kserve.io/port: "8000"
    prometheus.kserve.io/path: "/v1/metrics"
  name: llama-3-1-8b-instruct-1xgpu
  namespace: default
spec:
  predictor:
    tolerations:
      - key: nvidia.com/gpu
        operator: Exists
        effect: NoSchedule
    nodeSelector:

```

(continues on next page)

(continued from previous page)

```

nvidia.com/gpu: "true"
imagePullSecrets:
- name: edb-cred
model:
  modelFormat:
    name: nvidia-nim-llama-3.1-8b-instruct
  resources:
    limits:
      nvidia.com/gpu: "1"
    requests:
      nvidia.com/gpu: "1"
  runtime: nvidia-nim-llama-3.1-8b-instruct-1.3.3

```

Deploy the InferenceService:

```
kubectl apply -f InferenceService.yaml
```

3. Verify deployed models

List active InferenceServices:

```

kubectl get InferenceService \
- o custom-columns=NAME:.metadata.name,MODEL:.spec.predictor.model.modelFormat.name,
↪URL:.status.address.url,RUNTIME:.spec.predictor.model.runtime,GPU:.spec.predictor.
↪model.resources.limits.nvidia.com/gpu \
- -namespace=default
__OUTPUT__
NAME                                MODEL                                URL                                ↪
↪                                GPU                                RUNTIME                                ↪
llama-3-1-8b-instruct-1xgpu         nvidia-nim-llama-3.1-8b-instruct    http://llama-3-1-8b-
↪instruct-1xgpu.default.svc.cluster.local    nvidia-nim-llama-3.1-8b-
↪instruct-1.3.3    1

```

4. Retrieve runtime details

Check port and resources:

```

kubectl get ClusterServingRuntimes \
- o custom-columns=NAME:.metadata.name,IMAGE:.spec.containers[0].image,PORT:.spec.
↪containers[0].ports[0].containerPort,CPU:.spec.containers[0].resources.limits.cpu,
↪MEMORY:.spec.containers[0].resources.limits.memory \
- -namespace=all-namespaces

```

Next steps

- *How-To Monitor deployed models with KServe*
- *How-To Update GPU Resources for an InferenceService*

Related reading

- *Model Serving Concepts*
- *AI Factory terminology*
- *Model Serving in Hybrid Manager*

How to integrate with your app

Quickstart — Use an Inference Endpoint with Python

Use this quickstart to chat with a model deployed behind your private InferenceService (KServe). You pass the endpoint URL and access key, plus the model name. The example uses a production-ready, single-file Python script.

Time to complete: 5–10 minutes

Goals

- Call a private, governed model endpoint (OpenAI-compatible) over HTTP(s)
- Understand required headers and payload shape for chat completions
- Run a minimal demo script you can reuse in apps and CI/CD

Prerequisites

- An InferenceService is deployed and ready. See:
- *Model Serving Reference Manual*
- *How-To Create an InferenceService for NVIDIA NIM Container*
- External access path and access key, or an internal cluster-local path. See: - *Accessing KServe endpoints (internal and external)*
- Python 3.9+ and httpx installed: `pip install httpx`

Environment

Set these environment variables before running the script:

```
export EDB_API_URL="https://<portal>/inferenceservices/<inferenceservice-id>"
export EDB_API_TOKEN="<hm-user-access-key>"
export MODEL_NAME="meta/llama-3.1-8b-instruct" # or the model you serve
```

Notes:

- For internal callers, set EDB_BASE_URL to your cluster-local path (proxy or direct KServe URL).
- The endpoint path for chat is `/v1/chat/completions` appended to EDB_BASE_URL.

Run the demo script

Download and run the production-ready single file:

- Script: `hm_kservice_quickstart.py`

Example (custom prompt):


```
python hm_kserve_quickstart.py chat --prompt "Write a haiku about Postgres and GPUs."
```

Optional (use Hybrid Manager to list clusters and summarize):

```
pip install httpx typer
export EDB_API_URL="https://<hm-host>"; export EDB_API_TOKEN="<hm-access-token>";
↪export PROJECT_ID="<uuid>"
python hm_kserve_quickstart.py summarize-clusters --project-id "$PROJECT_ID"
```

Request and headers (reference)

- Endpoint: POST `${EDB_BASE_URL}/v1/chat/completions`
- Headers: Authorization: Bearer `${EDB_API_KEY}` , Accept: application/json , Content-Type: application/json
- Body (simplified):

```
{
  "model": "${MODEL_NAME}",
  "messages": [
    { "role": "user", "content": "Hello" }
  ],
  "max_tokens": 256
}
```

Embeddings and rerank models use different paths:

- Embeddings: `${EDB_BASE_URL}/v1/embeddings`
- Rerank: `${EDB_BASE_URL}/v1/ranking`

If you see HTTP 404 Not Found, verify the operation-specific path.

Best practices

- Keep inference sovereign: prefer internal paths when the caller runs in-cluster.
- Rotate access keys regularly; never commit them to source control.
- Enforce TLS and limit egress from clients.
- Monitor latency and error rates; scale resources or tune concurrency as needed.

Troubleshooting

- 401/403: Verify `EDB_API_KEY` and user permissions to the InferenceService.
- 404: Confirm the InferenceService ID and that the service is ready.
- Timeouts/5xx: Check KServe pod status, health probes, and logs; validate endpoint path.

Next steps

- Build an app using the same headers and payloads; see the [Quickstart — Use an Inference Endpoint with Python](#).
- Integrate with Gen AI Assistants and Knowledge Bases for RAG; see [Gen AI Builder](#) and Pipelines and data preparation.

- Add observability and SLOs; see *Observability for Model Serving* and *How-To Update GPU Resources for an InferenceService*.

Known issues in 1.3

- Some internal model URLs shown in listings may omit the operation suffix (for example, embeddings require `/v1/embeddings`). If a call returns 404, append the appropriate suffix. This will be addressed in a future release.
- External access requires a valid Hybrid Manager user access key with the right role (for example, Gen AI Builder User). A malformed key or insufficient permissions return HTTP 401.

hm_kserve_quickstart.py

```
import os, sys, json, httpx

BASE=os.getenv("EDB_API_URL","").rstrip("/")
TOKEN=os.getenv("EDB_API_TOKEN","")
ISVC=os.getenv("INFERENCESERVICE_ID","")
MODEL=os.getenv("MODEL_NAME","nvidia/llama-3.3-nemotron-super-49b-v1")
OPENAPI_FILE=os.getenv("OPENAPI_FILE","")

HEADERS={"Authorization":f"Bearer {TOKEN}", "Accept": "application/json", "Content-Type":
↪ "application/json"}

def load_docs():
    if not OPENAPI_FILE:
        print("warning: no OPENAPI_FILE provided; LLM has no API docs context")
        return "No API docs available."
    try:
        with open(OPENAPI_FILE, "r", encoding="utf-8") as f:
            spec=json.load(f)
            trimmed={"paths":spec.get("paths", {}), "tags":spec.get("tags", [])}
            return json.dumps(trimmed, ensure_ascii=False)[:6000]
    except Exception as e:
        print(f"warning: failed to read OPENAPI_FILE: {e}")
        return "No API docs available."

def chat(messages):
    body={"model":MODEL, "messages":messages, "max_tokens":4096}
    with httpx.Client(base_url=BASE, headers=HEADERS, timeout=60) as c:
        r=c.post(f"/inferenceservices/{ISVC}/v1/chat/completions", json=body)
        r.raise_for_status()
        return r.json()["choices"][0]["message"]["content"].strip()

def main():
    docs=load_docs()
    messages=[
        {"role": "system", "content": "You are an API helper for engineers."},
        {"role": "system", "content": "Here are the API docs:\n"+docs},
    ]
    print("API helper chat. Commands: /reset, /exit")
    try:
```

(continues on next page)

(continued from previous page)

```

while True:
    user=input("> ").strip()
    if not user: continue
    if user=="/exit": break
    if user=="/reset":
        messages=messages[:2]
        print("(reset to API docs)")
        continue
    messages.append({"role":"user","content":user})
    try:
        reply=chat(messages)
    except Exception as e:
        print(f"[error] {e}")
        messages.pop()
        continue
    messages.append({"role":"assistant","content":reply})
    print(reply)
except KeyboardInterrupt:
    print()

if __name__=="__main__":
    main()

```

Accessing KServe endpoints (internal and external)

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Use these steps to invoke model endpoints deployed with KServe.

Why and when to use this

Accessing KServe endpoints lets your applications consume private, governed model inference without depending on public AI APIs. You choose the access path that matches your security posture and network topology:

- Internal access: Use when callers run inside the same Kubernetes environment (for example, Gen AI Builder Assistants, Pipelines jobs). This path avoids external exposure, skips a gateway hop, and thus reduces latency.
- External access: Use when callers live outside the cluster (for example, customer-facing apps, partner services, or shared enterprise APIs). The portal with access keys provides a controlled boundary while keeping inference sovereign.

Typical reasons to use KServe access:

- Keep model inference in your environment for data protection, compliance, and cost control.
- Serve OpenAI-compatible endpoints to simplify client integration while retaining governance.
- Support air-gapped or restricted network environments where public AI services are not allowed.

Prerequisites

- A deployed InferenceService. See [How-To Create an InferenceService for NVIDIA NIM Container](#).
- For external access: a portal endpoint and a user access key from your Hybrid Manager environment.
- For internal access: network access to the service inside the Kubernetes cluster.

Internal access (inside the cluster)

Call the in-cluster address that fronts KServe (cluster-local DNS). Replace placeholders with your values.

Example (proxy service path style):

```
curl -X POST \
  http://upm-kserve-model-proxy.upm-ai-model-server.svc.cluster.local:80/
  ↪inferenceservices/<inferenceservice-id>/<endpoint-path> \
  -H accept: application/json \
  -H Content-Type: application/json \
  -d { ... }
```

Example (direct KServe service URL, when available in the resource status):

```
status:
  address:
    url: http://<inferenceservice-name>-predictor.<namespace>.svc.cluster.local
```

Use that URL in your HTTP client to invoke the model.

External access (outside the cluster)

Call the portal endpoint and include your Hybrid Manager user access key as a Bearer token. Replace placeholders with your values.

```
curl -X POST \
  https://<portal_domain>:<portal_port>/inferenceservices/<inferenceservice-id>/
  ↪<endpoint-path> \
  -H accept: application/json \
  -H Content-Type: application/json \
  -H "Authorization: Bearer a" \
  -d { ... }
```

Example (OpenAI-compatible chat completions):

```
curl -X POST \
  https://portal-HM-pm.edbHM.com/inferenceservices/m-po7d0fta/v1/chat/completions \
  -H accept: application/json \
  -H Content-Type: application/json \
  -H "Authorization: Bearer a" \
  -d {
    "model": "meta/llama-3.1-8b-instruct",
    "messages": [{"role": "user", "content": "Write a limerick about the wonders of GPU_
  ↪computing."}],
    "max_tokens": 64
  }
```

Internal vs. external URLs — where to find them

Use internal URLs for in-cluster callers and the external portal URL for callers outside Kubernetes.

- Internal URL (cluster-local):
- From SQL (AIDB): run `SELECT * FROM aidb.list_HM_models();` to see internal service URLs for known models within the database context.

- From Kubernetes: check the `status.address.url` on the `InferenceService` resource in the namespace (for example, `http://<name>-predictor.<ns>.svc.cluster.local`).
- Note: Some tasks require an operation suffix (for example, embeddings require `/v1/embeddings`). If a listed URL returns 404, append the proper suffix.
- External URL (portal): - In Hybrid Manager UI: open AI Factory → Model Serving → select your `InferenceService` → copy the portal endpoint (base) for the service. External calls use `https://<portal>/inferenceservices/<id>/...` with an HM user access key.

Endpoint paths by operation

KServe exposes different paths per operation. Use the path that matches your model/task:

- Chat completions (LLMs): `<base>/v1/chat/completions`
- Embeddings (NIM embedding models): `<base>/v1/embeddings`
- Rerank (ranking models): `<base>/v1/ranking`

If you see HTTP 404 Not Found when calling a model, verify that you're using the correct path for the operation.

Payloads by operation

Different tasks use different payload shapes. The examples below show common OpenAI-compatible schemas.

- Chat completions
- Endpoint: `<base>/v1/chat/completions`
- Payload:

```
{
  "model": "<model-name>",
  "messages": [
    { "role": "user", "content": "Hello" }
  ],
  "max_tokens": 256
}
```

- Embeddings
- Endpoint: `<base>/v1/embeddings`
- Payload:

```
{
  "model": "<model-name>",
  "input": "I am the Yonk, I work in product"
}
```

- Rerank
- Endpoint: `<base>/v1/ranking`
- Payload (example shape; check your model's schema):

```
{
  "model": "<model-name>",
  "query": "search phrase",
}
```

(continues on next page)

(continued from previous page)

```

"documents": ["doc one", "doc two"],
"top_n": 5
}

```

Notes:

- For AIDB SQL helpers (for example, `aidb.encode_text()`), the database will construct the HTTP call for you. If you configure a model manually with an internal URL, set the correct operation path (for example, `/v1/embeddings`) to avoid 404s.
- Some models expose additional operation paths; consult the model reference or your InferenceService documentation.

Security considerations

- Prefer internal access for services that do not need to be exposed externally.
- Rotate access keys regularly and scope them by user; do not embed keys in client code repositories.
- Enforce TLS for external calls; use certificates managed by your platform team.
- Limit egress from clients where possible; monitor usage via observability.

Troubleshooting

- 404 - Please check you are using the correct endpoints. (e.g. internal/external with matching id to model name)
- 401/403 on external calls: Verify the Authorization header and that the user has access to the InferenceService.
- Timeouts or 503: Check the InferenceService status, health probes, and backend logs; confirm that the endpoint path is correct.
- High latency: Right-size resources or adjust concurrency. See [How-To Update GPU Resources for an InferenceService](#).

Known issues in 1.3

- External access requires a valid Hybrid Manager user access key with sufficient role permissions (for example, Gen AI Builder User). Invalid or malformed keys return HTTP 401 Unauthorized.

Related

- [Model Serving Reference Manual](#)
- [How-To Create an InferenceService for NVIDIA NIM Container](#)
- [Observability for Model Serving](#)

How to operate and observe

How-To Verify InferenceServices and GPU usage

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See [/edb-postgres-ai/1.3/hybrid-manager/ai-factory/](#).

Use this guide to confirm the correct deployment and operational status of InferenceServices and GPU resource usage.

Goal

Ensure your deployed InferenceServices are correctly utilizing GPU resources.

Estimated time

5–10 minutes.

Steps

Check InferenceService status

```
kubectl get inferencservice -n <namespace>
```

Look for `READY` to ensure the service is running.

Confirm GPU resource usage

```
kubectl describe nodes | grep nvidia.com/gpu
kubectl exec -n <namespace> -it <pod-name> -- nvidia-smi
```

Troubleshoot common issues

```
kubectl get ds -n kube-system nvidia-device-plugin-daemonset
kubectl describe pods -n <namespace>
```

Related concepts

- [How-To Setup GPU resources](#)
- [How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe](#)
- [Model Serving Reference Manual](#)

How-To Monitor deployed models with KServe

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See [/edb-postgres-ai/1.3/hybrid-manager/ai-factory/](#).

This guide explains how to monitor deployed AI models using KServe on Kubernetes.

Monitoring your models helps ensure reliability, performance, and optimal use of resources — whether you are working in general Kubernetes or using Hybrid Manager AI Factory.

For AI Factory users, Hybrid Manager will provide additional value-add monitoring and observability features — see [Model Serving in Hybrid Manager](#).

Goal

Monitor deployed models, check model status and serving endpoints, and retrieve resource usage information.

Estimated time

5–10 minutes.

What you will accomplish

- List deployed InferenceServices (models).
- Retrieve model endpoint and runtime details.
- Understand how to observe model performance.
- Prepare to integrate model metrics into observability pipelines.

What this unlocks

- Confidence that models are correctly deployed and serving.
- Ability to troubleshoot or scale model deployments.
- Foundation for using Hybrid Manager AI Factory observability for model serving.

Prerequisites

- Deployed InferenceService on KServe.
- ClusterServingRuntime defined.
- kubectl configured for your Kubernetes cluster.

For background concepts, see:

- [Model Serving Concepts](#)
- [AI Factory terminology](#)

Steps

1. List deployed InferenceServices

To list deployed models and see key details:

```
kubectl get InferenceService \
- o custom-columns=NAME:.metadata.name,MODEL:.spec.predictor.model.modelFormat.name,
↪URL:.status.address.url,RUNTIME:.spec.predictor.model.runtime,GPU:.spec.predictor.
↪model.resources.limits.nvidia\.com/gpu \
- -namespace=default
```

Key columns:

- **NAME:** Name of the InferenceService.
- **MODEL:** Model format name (from ClusterServingRuntime).
- **URL:** Service endpoint for inference requests.
- **RUNTIME:** ClusterServingRuntime used.
- **GPUs:** Number of GPUs allocated.

2. Retrieve runtime details

To view ClusterServingRuntime details, including serving port and resource allocations:

```
kubectl get ClusterServingRuntimes \
- o custom-columns=NAME:.metadata.name,IMAGE:.spec.containers[0].image,PORT:.spec.
↪containers[0].ports[0].containerPort,CPU:.spec.containers[0].resources.limits.cpu,
↪MEMORY:.spec.containers[0].resources.limits.memory \
- --namespace=all-namespaces
```

Key columns:

- **NAME:** Name of the runtime.
- **IMAGE:** Model server image used.
- **PORT:** Inference port (commonly 8000 for NIM).
- **CPU:** CPU resources allocated.
- **MEMORY:** Memory allocated.

3. Observe model metrics

If you enabled Prometheus scraping via InferenceService annotations:

- serving.kserve.io/enable-prometheus-scraping: “true”

Then Prometheus can scrape metrics at:

/v1/metrics on port 8000 of the model service.

Metrics typically include:

- **Request latency**
- **Throughput (requests per second)**
- **Error rates**
- **GPU utilization (if GPUs used)**

You can visualize these metrics in tools such as **Grafana**.

4. Check pod status

For debugging, you can also view model pods directly:

```
kubectl get pods --namespace=default
```

Look for pods with names matching:

```
<inference-service-name>-predictor-*
```

Check pod status and logs if needed:

```
kubectl logs <pod-name> --namespace=default
```

Next steps

- *How-To Update GPU Resources for an InferenceService (Coming soon)*
- *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe*
- *Model Serving in Hybrid Manager (Coming soon)*

Related reading

- *Model Serving Concepts*
- *AI Factory terminology*
- *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe*

How-To Update GPU Resources for an InferenceService

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

This How-To explains how to adjust the number of GPUs allocated to an existing InferenceService deployed with KServe. This allows you to scale your model deployment dynamically without redeploying from scratch.

Goal

Change the GPU resource allocation (number of GPUs) for a deployed InferenceService.

Estimated time

5 minutes.

What you will accomplish

- Edit an existing InferenceService resource.
- Apply updated GPU resource limits and requests.
- Trigger KServe to redeploy the model container with new GPU settings.

What this unlocks

Enables dynamic scaling of GPU resources to optimize performance and cost:

- Increase GPUs for faster response time and higher throughput.
- Reduce GPUs when demand is lower to save resources.

Prerequisites

- An InferenceService already deployed using KServe. See *How-To Create an InferenceService for NVIDIA NIM Container*.
- GPU-enabled node pool available in your Kubernetes cluster.

Steps

1. Edit the InferenceService

Run:

```
kubectl edit InferenceService <your-inferenceservice-name>
```

Example:

```
kubectl edit InferenceService llama-3-1-8b-instruct-1xgpu-g5
```

2. Update GPU resource settings

Locate this section:

```
spec:
  predictor:
    model:
      resources:
    limits:
      nvidia.com/gpu: "1"
    requests:
      nvidia.com/gpu: "1"
```

Change both **limits** and **requests** to the desired number of GPUs. Example to scale to 4 GPUs:

```
spec:
  predictor:
    model:
      resources:
        limits:
          nvidia.com/gpu: "4"
        requests:
          nvidia.com/gpu: "4"
```

Save and close the editor.

3. Verify the updated GPU allocation

Run:

```
kubectl get InferenceService -o custom-columns=NAME:.metadata.name,MODEL:.spec.
↪predictor.model.modelFormat.name,URL:.status.address.url,RUNTIME:.spec.predictor.
↪model.runtime,GPU:.spec.predictor.model.resources.limits.nvidia.com/gpu --
↪namespace=default
```

Confirm that the **GPU**s column reflects your updated setting.

KServe will automatically redeploy the InferenceService with the new configuration.

Related topics

- [How-To Create an InferenceService for NVIDIA NIM Container](#)
- [How-To Configure a ClusterServingRuntime](#)
- [How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.comcontainer with KServe \(placeholder\)](#)
- [Gen AI Builder Concepts \(Assistants, Knowledge, Tools, Threads\)](#)
- [AI Factory terminology](#)

Next steps

- [How-To Monitor deployed models with KServe](#)
- [How-To Update GPU Resources for an InferenceService](#)
- [How-To Update GPU Resources for an InferenceService](#)

Explore more in the Analytics & AI Factory learning guide .

How-To Use NVIDIA NIM Model Cache in Air-Gapped Clusters in Hybrid Manager

Overview

Use a model cache to run NVIDIA NIM models in environments without internet access. The cache contains the model “profiles” files that the NIM container otherwise downloads from NVIDIA NGC at runtime.

This guide walks through a single flow for all cache types — you do not need separate pages per backend or object store.

- Phase 1: Build the model cache (connected environment)
- Phase 2: Upload the cache to object storage (AWS or GCP)
- Phase 3: Use the cache in your air-gapped cluster (UI only)

Prerequisites

- An HM cluster with GPU nodes. Label nodes `nvidia.com/gpu=true` and taint with `nvidia.com/gpu`.
- NVIDIA NGC API key.
- A private registry accessible from your air-gapped cluster.

Phase 1 — Build the model cache (connected)

In a connected environment, copy the NIM images you need to your private registry, discover compatible profiles, and pre-download them into a local cache directory.

What is a “model cache”?

- A model cache is a set of “profiles” files required by NVIDIA NIM images at runtime. NIM selects profiles based on hardware and backend and typically downloads them from NGC.
- Only NIM models can use this method. Other KServe-supported models cannot leverage the NIM profile cache.

1) Copy NIM images to your private registry

Use skopeo to copy images from NVIDIA NGC to your registry.

```
NGC_API_KEY=<your NGC API key>
REGISTRY=<your registry, e.g. registry.example.com>
USER=<your registry username>
PASSWORD=<your registry password>

skopeo login -u $oauthtoken -p "${NGC_API_KEY}" nvcr.io
skopeo login -u "${USER}" -p "${PASSWORD}" "${REGISTRY}"

# Example: text embeddings model

skopeo copy --override-os linux --multi-arch all \
  docker://nvcr.io/nim/nvidia/llama-3.2-nemoretriever-300m-embed-v1:latest \
  "docker://${REGISTRY}/nim/nvidia/llama-3.2-nemoretriever-300m-embed-v1:latest"
```

Repeat for each NIM image you plan to use. Common defaults in HM include:

```
nvcr.io/nim/nvidia/llama-3.3-nemotron-super-49b-v1:1.8.5
nvcr.io/nim/nvidia/llama-3.2-nemoretriever-300m-embed-v1:latest
nvcr.io/nim/nvidia/llama-3.2-nv-rerankqa-1b-v2:latest
nvcr.io/nim/nvidia/nvclip:latest
nvcr.io/nim/baidu/paddleocr:latest
```

2) (Optional) Update default model image URLs in HM

If HM is connected, update default model image URLs to point to your registry so future clusters pull from it.

```
access_key=<your HM API key>

# List models; note each model ID

curl -k -H "x-access-key:${access_key}" \
  -X GET "https://<your HM Portal URL>/api/v1/ai-models" | jq .

# Patch a models image URL

curl -k -H "x-access-key:${access_key}" \
  -X PATCH "https://<your HM Portal URL>/api/v1/ai-models/<model ID>" \
  -d {"imageUrl":"<your-registry>/nim/nvidia/llama-3.2-nemoretriever-300m-embed-
↪v1:latest"}
```

3) Discover compatible profiles

Profiles are model+hardware variants. Use the NIM container to list profiles for your target environment. You can do this directly with Docker on a GPU host:

```
export NGC_API_KEY=<your NGC API key>
docker run -it --rm \
  --runtime=nvidia \
  --gpus all \
```

(continues on next page)

(continued from previous page)

```
--shm-size=16GB \
-e NGC_API_KEY \
<your-registry>/nim/nvidia/llama-3.2-nemoretriever-300m-embed-v1:latest \
list-model-profiles
```

Record the profile IDs in the “Compatible with system” section for your GPUs.

Alternative: list profiles from Kubernetes (Job) If you cannot SSH to a GPU node, run a short-lived Job in a cluster with GPU access to print compatible profile IDs.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: nim-list-job
  namespace: default
spec:
  template:
    metadata:
      name: nim-list-pod
    spec:
      containers:
        - name: nim-list
          image: <your-registry>/nim/nvidia/llama-3.2-nemoretriever-300m-embed-
↪v1:latest
          args: ["list-model-profiles"]
          env:
            - name: NGC_API_KEY
              valueFrom:
                secretKeyRef:
                  name: nvidia-nim-secrets
                  key: NGC_API_KEY
      resources:
        limits:
          nvidia.com/gpu: "1"
        requests:
          nvidia.com/gpu: "1"
      tolerations:
        - key: nvidia.com/gpu
          operator: Exists
          effect: NoSchedule
      imagePullSecrets:
        - name: <your-pull-secret>
      restartPolicy: Never
      securityContext:
        fsGroup: 26
        runAsGroup: 26
        runAsNonRoot: true
        runAsUser: 26
        seccompProfile:
          type: RuntimeDefault
```

Example output (IDs vary by model and hardware):

MODEL PROFILES

- Compatible **with** system **and** runnable:
 - 4f904d571fe60ff24695b5ee2aa42da58cb460787a968f1e8a09f5a7e862728d (vllm-bf16-tp1-
→pp1)
- With LoRA support:
 - f749ba07aade1d9e1c36ca1b4d0b67949122bd825e8aa6a52909115888a34b95 (vllm-bf16-tp1-
→pp1-lora)
- Compilable to TRT-LLM using just-**in**-time compilation:
 - ac34857f8dcdb174ad524974248f2faf271bd2a0355643b2cf1490d0fe7787c2 (tensorrt_llm-
→trtllm_buildable-bf16-tp1-pp1)
- With LoRA support:
 - 7b8458eb682edb0d2a48b4019b098ba0bfbcb4377aadeeaa11b346c63c7adf724 (tensorrt_llm-
→trtllm_buildable-bf16-tp1-pp1-lora)
- Incompatible **with** system:
 - 6c3f01dd2b2a56e3e83f70522e4195d3f2add70b28680082204bbb9d6150eb04 (tensorrt_llm-
→h100-fp8-tp2-pp1-latency)

Note: Although HM doesn't currently expose LoRA support in the UI, you can still cache LoRA-capable profiles for manual deployments.

4) Build the cache locally

Download only the compatible profiles, or download all profiles if you prefer a universal cache.

Download selected profiles:

```
export NGC_API_KEY=<your NGC API key>
export LOCAL_NIM_CACHE=./model-cache
mkdir -p "$LOCAL_NIM_CACHE" && chmod -R a+w "$LOCAL_NIM_CACHE"

docker run -v $LOCAL_NIM_CACHE:/opt/nim/.cache \
  -u $(id -u) \
  -e NGC_API_KEY \
  --rm \
  <your-registry>/nim/nvidia/llama-3.2-nemoretriever-300m-embed-v1:latest \
  download-to-cache \
  --profiles \
  <profile_id_1> \
  <profile_id_2>
```

Download all profiles:

```
export NGC_API_KEY=<your NGC API key>
export LOCAL_NIM_CACHE=./model-cache
mkdir -p "$LOCAL_NIM_CACHE" && chmod -R a+w "$LOCAL_NIM_CACHE"

docker run -v $LOCAL_NIM_CACHE:/opt/nim/.cache \
  -u $(id -u) \
  -e NGC_API_KEY \
```

(continues on next page)

(continued from previous page)

```
--rm \
<your-registry>/nim/nvidia/llama-3.2-nemoretriever-300m-embed-v1:latest \
download-to-cache --all
```

After this step, `./model-cache` contains the profiles that the NIM container will use when offline. The folder layout looks like:

```
model-cache/
├─ nim-nvidia-llama-3.2-nemoretriever-300m-embed-v1
│   └─ local_cache
│       └─ ngc
│           └─ hub
│               └─ models--nim--nvidia--llama-3.2-nemoretriever-300m-embed-v1
│                   └─ blobs
│                       └─ <hash-1>
│                       └─ <hash-2>
│                       └─ ...
│                   └─ refs
│                       └─ <ref-name>
│                   └─ snapshots
│                       └─ <snapshot-name>
└─ tmp
```

Phase 2 — Upload the cache to object storage

Upload the local cache directory to your object store. Use a distinct prefix per model, for example `model-cache/nim-nvidia-llama-3.2-nemoretriever-300m-embed-v1`.

AWS S3 example

```
aws s3 cp -r ./model-cache \
s3://<bucket-name>/model-cache/nim-nvidia-llama-3.2-nemoretriever-300m-embed-v1
```

Optional: specify an AWS CLI profile

```
aws s3 cp -r ./model-cache \
s3://<bucket-name>/model-cache/nim-nvidia-llama-3.2-nemoretriever-300m-embed-v1 \
--profile <aws-profile>
```

GCP bucket example

```
gcloud storage cp -r ./model-cache \
gs://<bucket-name>/model-cache/nim-nvidia-llama-3.2-nemoretriever-300m-embed-v1
```

Supported object store URI formats

Hybrid Manager supports the following URI formats for the Model Profiles Path:

```
- s3://<bucket>/<path>
```

(continues on next page)

(continued from previous page)

```

- gs://<bucket>/<path>

- hdfs://<namenode-host>:<port>/<path>

- webhdfs://<namenode-host>:<port>/<path>

- https://<account>.blob.core.windows.net/<container>/<path> (Azure Blob Storage)

- https://<hostname>/<path> (generic HTTPS object stores)

```

Phase 3 — Use the cache in an air-gapped cluster

Create your model from the HM UI and point it at the cache path on your object store. No Kubernetes YAML is required (this differs from KServe).

Steps in HM UI

1. Go to AI Factory → Models → Create Model
2. Choose a default model (for example: llama-3.2-nemoretriever-300m-embed-v1)
3. Set “Model Profiles Path on Object Storage” to the path you uploaded in Phase 2, for example:
 - s3://<bucket-name>/model-cache/nim-nvidia-llama-3.2-nemoretriever-300m-embed-v1
 - gs://<bucket-name>/model-cache/nim-nvidia-llama-3.2-nemoretriever-300m-embed-v1
4. Select your compute (GPU) and size; then create

Screenshots

- Screenshot placeholder: Create Model — Model selection
- Screenshot placeholder: Create Model — Model Profiles Path
- Screenshot placeholder: Model detail — Cache in use

Notes

- Ensure your cluster can pull images from your private registry (configure image pull secrets in HM or the cluster as needed).
- Build and upload caches for each model you plan to use.
- You do not need to create secrets or apply any YAML to use the cache in HM.

Validate the cache and deployment

Follow these steps to verify the image copy, cache upload, and that your model is using the cache in the air-gapped environment.

1. Validate images in your private registry

```

skopeo inspect docker://${REGISTRY}/nim/nvidia/llama-3.2-nemoretriever-300m-embed-
v1:latest | jq .Name, .Digest

```

2. Validate the local cache before upload

```
du -sh ./model-cache
find ./model-cache -maxdepth 3 -type d | head -n 20
```

3. Validate the object store upload

- **S3:** `aws s3 ls s3://<bucket-name>/model-cache/nim-nvidia-llama-3.2-nemoretriever-300m-embed-v1/`
- **GCS:** `gcloud storage ls gs://<bucket-name>/model-cache/nim-nvidia-llama-3.2-nemoretriever-300m-embed-v1/`
- **HDFS:** `hdfs dfs -ls /model-cache/nim-nvidia-llama-3.2-nemoretriever-300m-embed-v1`
- **WebHDFS:** `curl -sS "http://<namenode-host>:9870/webhdfs/v1/model-cache/nim-nvidia-llama-3.2-nemoretriever-300m-embed-v1?op=LISTSTATUS" | jq`

4. Validate in HM UI

- The model card shows status Ready/Healthy
 - The “Model Profiles Path on Object Storage” matches your S3/GS/HDFS/WebHDFS/HTTPS path
5. Run a smoke test against the model endpoint From the model details page, copy the endpoint URL and API key. Then, send a minimal embeddings request (for the embeddings model example):

```
ENDPOINT=<your model base URL>
API_KEY=<your model api key>

curl -sS -X POST "$ENDPOINT/v1/embeddings" \
  -H "Authorization: Bearer $API_KEY" \
  -H "Content-Type: application/json" \
  -d {"input":["hello from EDB"]} | jq .data[0].embedding | length
```

You should see a numeric vector length returned.

6. (Optional) Inspect the running pod’s cache usage

```
kubectl -n <model-namespace> get pods -l app.kubernetes.io/name=<model-name>
kubectl -n <model-namespace> logs <pod-name> --tail=200
kubectl -n <model-namespace> exec -it <pod-name> -- sh -lc ls -al /opt/nim/.cache &&
↪du -sh /opt/nim/.cache
```

Logs should not show attempts to download from NGC when the cache is present.

3.3 Gen AI Builder

3.4 Gen AI Builder

Gen AI Builder provides comprehensive tooling for developing intelligent applications that leverage organizational knowledge through *retrieval* *augmented generation* patterns. The platform operates within controlled infrastructure environments, ensuring complete organizational sovereignty over data, models, and inference operations.

New here? Start with *Gen AI Builder Hub* , review *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)* and the *Gen AI Architecture (Entities and Flows)* , then use the *Gen AI How-To Guides* or jump into the *Customer Service Assistant Implementation Guide* .

3.4.1 Platform Architecture

Gen AI Builder integrates multiple components to deliver end-to-end AI application development capabilities within the EDB Postgres AI ecosystem. The platform emphasizes data sovereignty by maintaining all operations within organizational boundaries while providing enterprise-grade governance and auditability. Explore the *Assistants Reference Manual*, *Knowledge Bases Reference Manual*, *Retrievers Reference Manual*, *Rulesets Manual*, *Structures Reference Manual*, *Tools Reference Manual*, *Threads Reference Manual*, *Configure Data Sources in Gen AI Builder*, *Data Lake Reference Manual*, and *LibrariesΩ Manual* manuals.

Core Infrastructure Integration

Hybrid Manager Foundation Gen AI Builder operates within *EDB's vision for AI with Postgres and Hybrid Manager*'s Kubernetes orchestration layer, leveraging distributed computing capabilities while maintaining centralized governance. This architecture ensures consistent operational procedures across development and production environments.

Data Estate Integration *Knowledge Bases Reference Manual* connect directly to Postgres databases and object storage systems (see *Data Lake Reference Manual*) within organizational infrastructure, eliminating external data dependencies while supporting diverse *Configure Data Sources in Gen AI Builder* types.

Model Serving Integration Applications access language models and embedding models through the *AI Factory Models* infrastructure, ensuring consistent governance policies across all AI capabilities.

3.4.2 Architectural Components

Knowledge Base Framework

Knowledge Bases Reference Manual serve as the foundation for retrieval-augmented generation applications by combining structured database content with unstructured document repositories. The framework provides semantic search capabilities through vector embeddings (see *Vector Engine (PGvector)*) while maintaining complete data lineage.

Data Source Integration

- Postgres database tables with semantic indexing
- Document repositories from object storage systems
- Real-time data feeds through API integrations
- Web content through controlled scraping operations

Embedding Management

- Private embedding models for sensitive content processing
- Vector storage optimization within Postgres infrastructure
- Semantic similarity algorithms for retrieval accuracy
- Hybrid search combining vector and keyword approaches

Retrieval Systems

Retrievers Reference Manual components orchestrate search operations across diverse data sources, providing unified interfaces for knowledge discovery while maintaining source attribution and access controls.

Retrieval Strategies

- Vector similarity search for semantic content matching
- Keyword-based search for exact term identification
- Hybrid approaches combining multiple retrieval methods
- Contextual filtering based on user permissions and data classification

Assistant Framework

Intelligent *Assistants Reference Manual* combine knowledge retrieval with language model capabilities to provide conversational interfaces for organizational knowledge. The framework supports complex reasoning workflows while maintaining audit trails for all interactions (see *Threads Reference Manual*).

Assistant Capabilities

- Multi-turn conversational interfaces with context preservation
- *Tools Reference Manual* integration for external system interactions
- Reasoning workflows for complex query resolution
- Audit logging for compliance and operational transparency

3.4.3 Implementation Patterns

Enterprise Knowledge Systems

Organizations implement comprehensive knowledge management systems that unify structured and unstructured information sources through semantic search capabilities.

Typical Architecture

1. **Data Integration Layer:** Connects multiple organizational *Configure Data Sources in Gen AI Builder* including databases, document repositories, and knowledge management systems
2. **Processing Pipeline:** Transforms content into searchable formats with appropriate metadata and access control annotations
3. **Retrieval Interface:** Provides unified search across all integrated content with relevance ranking and source attribution
4. **Application Layer:** Delivers knowledge access through conversational interfaces and API integrations

Implementation Considerations

- Data governance policies for content classification and access control
- Performance optimization for large document collections
- Integration with existing authentication and authorization systems
- Compliance requirements for audit trails and data lineage

Retrieval-Augmented Generation Applications

Applications combine retrieval capabilities with language model inference to provide context-aware responses based on organizational knowledge.

Development Workflow

1. **Knowledge Base Construction:** Establish comprehensive document indexing with semantic embeddings (see *Create a Knowledge Base in Gen AI Builder* and *Manage Knowledge Bases in Gen AI Builder*)
2. **Retrieval Configuration:** Implement search strategies optimized for application requirements
3. **Generation Integration:** Connect language models for response generation based on retrieved context
4. **Response Orchestration:** Coordinate retrieval and generation workflows with appropriate caching and optimization

Technical Requirements

- Low-latency retrieval for interactive applications

- Context window management for large document processing
- Response quality evaluation and improvement workflows
- Integration testing across knowledge base updates

Conversational AI Platforms

Interactive applications provide natural language interfaces for complex organizational workflows and information discovery.

Platform Components

- Multi-modal input processing for text, voice, and document interactions
- Context management for extended conversational sessions
- Tool integration for external system operations
- Response personalization based on user roles and preferences

Operational Features

- Session management with conversation state persistence
- Performance monitoring for response quality and latency
- Usage analytics for application optimization and capacity planning
- Security controls for sensitive information handling

3.4.4 Development Framework

Knowledge Base Development

Knowledge base creation involves data source integration, content processing, and semantic indexing to support efficient retrieval operations.

Development Process

1. **Source Assessment:** Evaluate *Configure Data Sources in Gen AI Builder* for content quality, update frequency, and access requirements
2. **Processing Pipeline Design:** Configure content transformation workflows including cleaning, chunking, and metadata extraction (see *Configure the Data Lake in Gen AI Builder*)
3. **Indexing Strategy:** Implement embedding generation and vector storage optimization
4. **Retrieval Testing:** Validate search quality through systematic evaluation procedures

Quality Assurance

- Content accuracy verification through systematic sampling
- Retrieval relevance evaluation using standard metrics
- Performance benchmarking under expected load conditions
- Security testing for access control and data isolation

Assistant Configuration

Assistant development requires careful orchestration of retrieval, reasoning, and response generation capabilities to deliver reliable user experiences (see *Create an Assistant in Gen AI Builder*).

Configuration Areas

- **Behavior Definition:** Establish assistant personality, capabilities, and operational boundaries
- **Knowledge Integration:** Configure retrieval strategies and knowledge source prioritization (see *Retrievers Reference Manual*)
- **Tool Integration:** Connect external systems and APIs for expanded functionality (see *Tools Reference Manual*)
- **Response Optimization:** Tune generation parameters for appropriate response characteristics

Testing Strategies

- Conversational flow validation through systematic dialog testing
- Knowledge accuracy verification across diverse query types
- Integration testing for external tool interactions
- Load testing for concurrent user scenarios

API Development

Applications expose functionality through standardized APIs that support diverse client applications while maintaining consistent security and operational characteristics (see *Where to find SDK references and examples?*).

API Design Principles

- RESTful interfaces with clear resource modeling
- Authentication and authorization integration with organizational systems
- Rate limiting and quota management for resource protection
- Comprehensive documentation and SDK support

3.4.5 Integration Architecture

EDB Postgres AI Ecosystem

Gen AI Builder integrates seamlessly with other EDB Postgres AI components to provide comprehensive AI application development capabilities.

Component Interactions

- **** AI Factory Models **:** Governed access to language models and embedding models
- **** AI Accelerator Pipelines **:** Automated content processing and knowledge base updates
- **** Vector Engine (PGvector) **:** High performance semantic search within Postgres infrastructure
- **** EDB's vision for AI with Postgres and Hybrid Manager **:** Orchestration and resource management

External System Integration

Applications connect with existing organizational systems through standardized interfaces while maintaining security boundaries and audit capabilities.

Integration Patterns

- API-based connections for real-time data access
- Batch processing for large data source synchronization
- Event-driven updates for dynamic content management
- Secure authentication delegation for user access control

3.4.6 Operational Considerations

Performance Characteristics

Gen AI Builder applications require careful performance optimization to deliver responsive user experiences while managing computational resources efficiently.

Performance Factors

- Knowledge base size affects retrieval latency and resource requirements
- Model complexity influences response generation time and GPU utilization
- Concurrent user load impacts system scalability and resource allocation
- Network connectivity affects distributed component communication

Security Framework

Comprehensive security controls protect sensitive organizational knowledge while enabling appropriate access for authorized users and applications.

Security Components

- Multi-tenant isolation for different organizational units
- Role-based access control aligned with existing authorization systems
- Data encryption for knowledge base content and communication channels
- Audit logging for all knowledge access and assistant interactions

Governance Integration

Built-in governance capabilities ensure applications comply with organizational policies and regulatory requirements throughout their operational lifecycle.

Governance Features

- Content approval workflows for knowledge base updates
- Model usage tracking and compliance reporting
- Data lineage maintenance for audit and troubleshooting
- Performance monitoring aligned with service level objectives

3.4.7 Getting Started

Prerequisites

Gen AI Builder requires foundational infrastructure components including Hybrid Manager installation and appropriate compute resources for AI workloads.

Infrastructure Requirements

- Kubernetes cluster with sufficient compute and storage capacity
- GPU resources for model serving and embedding generation
- Network connectivity to organizational data sources
- Integration with identity and access management systems

Initial Implementation

Organizations should begin with focused use cases that demonstrate value while establishing operational procedures for broader deployment.

Implementation Sequence

1. **Pilot Knowledge Base:** Create initial knowledge base with well-defined content scope
2. **Basic Retrieval:** Implement simple search functionality with performance validation
3. **Assistant Prototype:** Develop conversational interface for pilot knowledge domain
4. **Production Deployment:** Scale successful prototypes with comprehensive operational controls

Learning Resources

Conceptual Foundation

- *Gen AI Builder Hub* - Platform overview and initial setup procedures
- *Gen AI How-To Guides* - Practical implementation guidance for common scenarios

Component Documentation

- *Libraries Manual* - Comprehensive knowledge base development guidance
- *Assistants Reference Manual* - Intelligent assistant creation and configuration
- *API Reference* - Detailed technical documentation for development teams

Advanced Topics

- *Data Lake Reference Manual* - Large-scale data processing and management
- *SDK Documentation* - Programmatic interfaces for custom development

Gen AI Builder enables organizations to develop sophisticated AI applications while maintaining complete control over data, models, and inference operations within their existing infrastructure environments.

Getting Started

Gen AI Builder Hub

Gen AI Builder is the developer-facing environment of **AI Factory**, designed to help you build advanced agentic AI applications on trusted data and governed infrastructure.

Builder gives you fine-grained control over **Knowledge Bases**, **Retrievers**, **Rulesets**, **Structures**, and **Tools** — letting you compose AI-powered applications that integrate deeply with your business context.

Gen AI Builder is the foundation for building Sovereign AI agents and data-powered AI workflows in AI Factory.

Core Capabilities

- Manage **Knowledge Bases** and power RAG-based experiences
- Create and manage **Retrievers** for optimized semantic and structured search
- Define **Rulesets** to control assistant tone, behavior, and compliance
- Build and deploy **Tools** to extend agent capabilities with API calls, calculations, and more

- Package and run **Structures** — complex Griptape-powered workflows and data pipelines
- Monitor and analyze **Threads** — conversation history and system behavior

Learning Paths

Start here to learn Builder step-by-step:

- *Getting Started - AI Factory 101*
- *Building Production-Ready AI Features - AI Factory 201*
- *Advanced AI Factory Usage and Extensibility - AI Factory 301*

Concepts and Explained

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Assistants Reference Manual*
- *Knowledge Bases Reference Manual*
- *Retrievers Reference Manual*
- *Rulesets Manual*
- *Tools Reference Manual*
- *Structures Reference Manual*
- *Threads Reference Manual*

How-To Guides

- *Create a Knowledge Base in Gen AI Builder*
- *Create a Retriever in Gen AI Builder*
- *Create a Ruleset in Gen AI Builder*
- *Create a Tool in Gen AI Builder*
- *Create a Structure in Gen AI Builder*
- *How-To View and Manage Threads in Gen AI Builder in Hybrid Manager*

Reference

- Gen AI Builder SDK reference

Related Hub Pages

- Gen AI in Hybrid Manager
- *Model Serving Reference Manual*
- Vector Engine Hub
- *AI Factory Hub Map*

When to use Gen AI Builder

Use Gen AI Builder when:

- You need **fine-grained control** of what your assistants retrieve and how they behave.
- You want to integrate **structured and unstructured data** into your agents.
- You need to build **trusted, compliant, production-grade agents** — not simple chatbot demos.
- You want to **extend agents with Tools** that connect to your real-world systems.
- You are building **multi-step AI pipelines and workflows** using Structures.

AI Factory Gen AI Builder gives you the tools to build advanced, extensible, enterprise-ready AI applications — entirely under your governance and with full visibility.

Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)

This page introduces the core entities you use to build Gen AI applications in AI Factory and how they relate. It orients you before you dive into how `tos` or SDK details.

Assistant (orchestrator)

An Assistant is the orchestrator of your application logic:

- Executes the directive (goal) and coordinates retrieval and tool use.
- Attaches to Knowledge (for RAG) and uses Tools (for actions, APIs).
- Runs with a model endpoint configured by your platform (see *Model Serving Reference Manual*).

Learn more: *Assistants Reference Manual*

Ruleset (policy and constraints)

Rulesets constrain behavior and guide output:

- Capture policy: tone, style, safety guidelines, guardrails.
- Apply to an Assistant to enforce boundaries across runs.

Learn more: *Rulesets Manual*

Knowledge (retrieval)

Your knowledge stack grounds the model on enterprise data:

- Knowledge Bases: embedded content from tables/files with metadata for filters.
- Retrievers: define search parameters (top□K, thresholds, filters) for hybrid search.
- Data Lake: repository for assets and files that feed knowledge ingestion.

Learn more:

- [Knowledge Bases Reference Manual](#)
- [Data Lake Reference Manual](#)

Tools (actions and integrations)

Tools allow Assistants to call external systems and internal APIs:

- Wrap REST endpoints or internal services with a typed interface.
- Provide authentication and input/output mapping.
- Govern which tools an Assistant can call via Rulesets and configuration.

Learn more: [Tools Reference Manual](#)

Structures (workflows and building blocks)

Structures are higher□level building blocks (tasks, pipelines, workflows) you can compose:

- Encapsulate reusable logic patterns.
- Integrate with Assistants or call Tools and retrieval internally.

Learn more: [Structures Reference Manual](#)

Threads (state and observability)

Threads persist conversation state and provide auditability:

- Record prompts, tool calls, retrieved context, and responses.
- Enable review, debugging, and compliance.

Learn more: [Threads Reference Manual](#)

Models and endpoints (generation)

Assistants use model endpoints for generation:

- Use [Model Serving Reference Manual](#) for private inference.
- Call endpoints internally (cluster□local) or externally (portal + access key).
- Use operation□specific paths: chat (/v1/chat/completions), embeddings (/v1/embeddings), rerank (/v1/ranking).

Learn more: [Accessing KServe endpoints \(internal and external\)](#)

Putting it together

At a high level:

- 1) Prepare Knowledge (knowledge bases with hybrid search) and Tools. 2) Configure an Assistant with a directive, attach Knowledge and Tools, and select a model endpoint. 3) Use Threads to observe behavior, debug traces, and govern interactions.

Build guides:

- [Customer Service Assistant Implementation Guide](#)
- [Gen AI Builder Hub](#)

Gen AI Architecture (Entities and Flows)

This page describes how Gen AI entities work together at runtime and where configuration and governance live. Use it as a companion to concepts, how-tos, and quickstarts.

Entities and relationships

- Assistant → orchestrates retrieval and tool use to fulfill the directive.
- Ruleset → constrains Assistant behavior (policy, tone, safety).
- Knowledge → knowledge bases (embedded content) + retrievers (hybrid search) + data lake inputs.
- Tools → enable actions and integrations via authenticated, typed interfaces.
- Structures → reusable building blocks (tasks, pipelines, workflows) called by Assistants.
- Threads → capture state and execution trace (prompts, retrievals, tool calls, outputs).

Entity docs:

- [Assistants Reference Manual](#)
- [Rulesets Manual](#)
- [Knowledge Bases Reference Manual](#)
- [Tools Reference Manual](#)
- [Structures Reference Manual](#)
- [Threads Reference Manual](#)

Control and data flows

```
User/App → Assistant → (Retriever → Knowledge Base) → Model endpoint
                      ↳ (Tool call → API/service)
                      ↳ Thread (state, trace)
```

Flow

1. User or app calls the Assistant with a directive/input.
2. Assistant retrieves context via the configured retriever and knowledge base(s).
3. Assistant optionally invokes Tools to gather data or perform actions.
4. Assistant calls the model endpoint for generation, grounded on retrieved context.
5. Thread captures prompts, retrieved chunks, tool I/O, and responses for audit and debugging.

Endpoints and models

- Use private *Model Serving Reference Manual* endpoints for Sovereign AI.
- Internal calls use cluster-internal DNS; external calls use portal + access keys.
- Operation paths: chat (`/v1/chat/completions`), embeddings (`/v1/embeddings`), rerank (`/v1/ranking`).
- Client patterns: OpenAI-compatible or raw HTTP; see *Accessing KServe endpoints (internal and external)* and *Quickstart — Use an Inference Endpoint with Python*.

Governance and observability

- Governing inputs: knowledge curation, retriever filters, tool scopes, rulesets.
- Observability: use Threads and platform metrics/logs to debug and monitor.
- Auditability: Threads provide a durable record of decisions and data use.

Where to start

- *Customer Service Assistant Implementation Guide*
- *Gen AI Builder Hub*
- *Best practices for Hybrid Knowledge Bases*
- *Create a Tool in Gen AI Builder*
- *How-To View and Manage Threads in Gen AI Builder in Hybrid Manager*

Gen AI Builder FAQ

This FAQ focuses on Gen AI Builder inside Hybrid Manager’s AI Factory. It covers common questions about models, knowledge bases, assistants, tools, SDK usage, security, performance, and operations.

Table of contents

- *Typical use cases and patterns*
- *Which models/drivers are supported?*
- *When to use RAG vs. fine-tuning?*
- *How do I create and update Knowledge Bases?*
- *What are Retrievers and how do I tune them?*
- *How do Assistants call Tools and Structures?*
- *Where does data live and how is access governed?*
- *How do I improve latency and control cost?*
- *How do I test, observe, and troubleshoot?*
- *Where to find SDK references and examples?*

Which models/drivers are supported?

Gen AI Builder is driver-based. You can use private/endpoints or hosted APIs via Prompt Drivers, Embedding Drivers, and Vector Store Drivers.

- Prompt Drivers: OpenAI, Anthropic, Google, and more (see reference: [reference/sdk/drivers/index.mdx](#) and per-provider pages)
- Embedding Drivers: OpenAI, Google, NVIDIA NIM, Hugging Face, etc. ([reference/sdk/drivers/embedding-drivers.mdx](#))
- Vector Store Drivers: pgvector (Postgres), local, Pinecone, Qdrant, Redis, and others ([reference/sdk/drivers/vector-store-drivers.mdx](#))
- NVIDIA NIM: deploy as private endpoints via Model serving and inference

See also: *AI Factory Models* and *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com container with KServe*.

When to use RAG vs fine-tuning?

Prefer RAG for living knowledge and governed answers; use fine-tuning for tone/format/narrow tasks. Many production assistants combine both: retrieve context from *Knowledge Bases Reference Manual* and guide responses with *Rulesets Manual*. For modular pipelines, explore RAG Engines.

How do I create and update Knowledge Bases?

- Create: *Create a Knowledge Base in Gen AI Builder* and configure *Configure Data Sources in Gen AI Builder* (Confluence, Google Drive, S3, Web Page, Data Lake, Custom)
- Update: *Manage Knowledge Bases in Gen AI Builder*; re-sync when source Libraries change
- Storage: embeddings live in Postgres (pgvector) or a configured vector store
- Tuning: choose chunking and metadata that match your retrieval; validate with golden questions

SDK references: [Data Loaders](#), [Embedding Drivers](#), [Vector Store Drivers](#).

What are Retrievers and how do I tune them?

Retrievers control how assistants fetch context: target KBs, max tokens, filters, re-ranking.

- Create: *Create a Retriever in Gen AI Builder*
- Tune: similarity thresholds, top-K, metadata filters, rerankers
- Advanced: modular RAG Engines with retrieval/rerank/response stages

SDK references: [RAG Engines](#), [Rerank Drivers](#).

How do Assistants call Tools and Structures?

Assistants orchestrate retrieval + generation + actions. Use Tools for external systems, or promote Structures (pipelines/workflows/agents) as callable Tools.

- Build assistants: *Create an Assistant in Gen AI Builder* with *Create a Ruleset in Gen AI Builder*, *Create a Retriever in Gen AI Builder*, and *Create a Tool in Gen AI Builder*
- Structures: package logic as a ZIP and deploy *Create a Structure in Gen AI Builder*; expose as Tool or run standalone
- Memory and threads: see *Threads Reference Manual* and conversation memory drivers

SDK references: *Structures Reference Manual* , *Tools Reference Manual* , Assistant Drivers , Conversation Memory Drivers .

Where does data live and how is access governed?

- Documents: ingest via *Configure Data Sources in Gen AI Builder* ; store in governed Data Lake or Postgres
- Embeddings/vectors: Postgres pgvector (recommended) or another configured vector store
- Governance: enforce permissions in source systems; restrict tool usage per project; audit retrieval and tool calls with threads/logs

See: *Configure the Data Lake in Gen AI Builder* , *Vector Engine (PGvector)* .

How do I improve latency and control cost?

- Retrieval: reduce top-K, improve chunking/metadata, use re-rank selectively
- Generation: choose right model per route; stream responses; batch where safe
- Caching: memoize embeddings, hot retrievals, and tool outputs when possible
- Infra: colocate models and KBs; scale with Model Serving autoscaling

See SDK: Engines and Drivers. See Models: Model serving and inference .

How do I test, observe, and troubleshoot?

- Testing: golden sets, conversation playbooks, SDK unit tests
- Observability: thread logs, assistant/structure run events; optionally export to your observability stack
- Debugging: verify retrieval set first; inspect ruleset changes; re-run the same structure/assistant with stored inputs

Docs: *Threads Reference Manual* , *SDK Observability for Model Serving* .

Typical use cases and patterns

- Enterprise knowledge assistants: KB + Retriever + Assistant + Tooling (tickets/CRM)
- Customer support copilots: policy/FAQ KBs + routing + guardrails via Rulesets
- Workflow bots: Structures + Tools for approvals, data enrichment, and reporting
- RAG for analytics: pgvector + Pipelines + Assistant for guided exploration

Explore: *Best practices for Hybrid Knowledge Bases* , *Customer Service Assistant Implementation Guide* .

Where to find SDK references and examples?

- SDK overview: Gen AI Builder SDK
- Data: Artifacts , Loaders , Chunkers
- Drivers: *Prompt/Assistant* , *Embeddings and vector representations* , Vector Stores
- Structures: *Tasks* , Pipelines and data preparation , *Structures (workflows and building blocks)*
- Tools: *Tools Reference Manual*

See also the product guides: *Assistants Reference Manual* , *Knowledge Bases Reference Manual* , *Retrievers Reference Manual* , *Rulesets Manual* , *Structures Reference Manual* , *Tools Reference Manual* , *Threads Reference Manual* .

Manual

Assistants Reference Manual

Assistants (Gen AI Builder, powered by Griptape) serve as the orchestration layer for intelligent applications, combining knowledge retrieval, behavioral governance, and tool execution into conversational interfaces. They transform static *Knowledge Bases Reference Manual* and *Tools Reference Manual* into interactive, governed AI experiences that run inside *EDB's vision for AI with Postgres and Hybrid Manager*.

Table of contents

- *Getting Started - AI Factory 101*
- *Practical Scenarios*
- *Configuration Checklist*
- *Architectural Role*
- *Core Architecture*
- *Operational Characteristics*
- *Integration Ecosystem*
- *Use cases*
- *SDK Quick Links*
- *FAQ*

Getting Started

- Create an assistant: *Create an Assistant in Gen AI Builder*
- Prepare data: *Create a Knowledge Base in Gen AI Builder*, *Manage Knowledge Bases in Gen AI Builder*
- Control retrieval: *Create a Retriever in Gen AI Builder*
- Define behavior: *Create a Ruleset in Gen AI Builder*
- Add actions: *Create a Tool in Gen AI Builder* or *Create a Structure in Gen AI Builder* and expose it as a tool

Typical flow: Knowledge Base → Retriever → Ruleset → Tools/Structures → Assistant → *Threads Reference Manual*

Practical Scenarios

- Enterprise knowledge assistant: Answer employee questions from trusted documentation with citations.
- Build steps: *Create a Knowledge Base in Gen AI Builder* → *Create a Retriever in Gen AI Builder* → *Create a Ruleset in Gen AI Builder* → *Create an Assistant in Gen AI Builder*
- Optional: add *Tools Reference Manual* for ticket creation or document lookup; review in *Threads Reference Manual*.
- Customer support copilot: Guide agents with policy/FAQ grounding and CRM actions.
- Build steps: KB of policies/FAQs → *Retrievers Reference Manual* tuned for recency → *Rulesets Manual* guardrails → *Create an Assistant in Gen AI Builder*
- Actions: *Create a Tool in Gen AI Builder* for CRM/ticketing; enforce scope via Rulesets.
- Workflow bot: Automate approvals and data enrichment with governed actions.
- Build steps: package business logic as a *Create a Structure in Gen AI Builder* and expose it as a Tool → attach to *Create an Assistant in Gen AI Builder*

- Logging: verify execution and outcomes in *Threads Reference Manual* .
- RAG for analytics: Help users explore data with pgvector-backed retrieval.
- Build steps: prepare embeddings with Pipelines and data preparation or *Configure the Data Lake in Gen AI Builder* + KB → *Retrievers Reference Manual* → *Create an Assistant in Gen AI Builder*
- Tune: chunking/metadata, similarity thresholds, and rerank modules.

Configuration Checklist

- Model/driver selection appropriate to task (see SDK *Which models/drivers are supported?*)
- Knowledge: one or more *Knowledge Bases Reference Manual* and a tuned *Retrievers Reference Manual*
- Behavior: *Rulesets Manual* with clear, non-conflicting guidance
- Actions: *Tools Reference Manual* and/or *Structures Reference Manual* with scoped credentials
- Memory/Threads: enable auditing, observability, and review flows (*Threads Reference Manual*)

Architectural Role

Assistants function as the primary interface between users and the Gen AI Builder ecosystem, coordinating multiple components to deliver coherent AI-powered interactions. They integrate with other AI Factory components including Model serving and inference for inference capabilities, *AI Accelerator Pipelines* for data processing, and *Vector Engine (PGvector)* for semantic search operations.

System Integration

Gen AI Builder Components

- **Knowledge Bases:** Provide factual grounding through retrieval-augmented generation patterns
- **Retrievers:** Execute semantic search operations across organizational knowledge
- **Rulesets:** Define behavioral parameters and operational boundaries
- **Tools:** Enable external system integration and action execution
- **Threads:** Maintain conversation state and interaction history

AI Factory Infrastructure

- **Model Serving:** Supplies language model inference capabilities through governed endpoints
- **Hybrid Manager:** Provides underlying orchestration and resource management
- **Vector Engine:** Delivers high-performance semantic search within Postgres infrastructure

Core Architecture

Conversation Processing Pipeline

Assistant operation follows a structured processing sequence that ensures consistent behavior and comprehensive knowledge utilization:

1. **Input Analysis:** User queries undergo parsing and intent classification
2. **Knowledge Retrieval:** Relevant information extraction from connected knowledge bases
3. **Behavioral Application:** Ruleset guidelines applied to response generation parameters
4. **Tool Execution:** External actions performed when appropriate and authorized

5. **Response Generation:** Language model inference with retrieved context and behavioral constraints
6. **Output Formatting:** Response packaging with citations and source attribution

Component Orchestration

Knowledge Integration Assistants coordinate multiple knowledge bases through *Retrievers Reference Manual* configurations that optimize search strategies based on content types and organizational requirements. This architecture supports hybrid search approaches combining vector similarity with keyword matching.

Behavioral Governance Rulesets provide operational boundaries that ensure assistant responses align with organizational policies, compliance requirements, and brand standards. These guidelines apply consistently across all interactions while maintaining conversational flexibility.

Tool Integration External system connectivity enables assistants to perform actions beyond information retrieval, including data updates, workflow initiation, and third-party service integration. Tool execution maintains audit trails and access control enforcement. See *Tools Reference Manual* and *Structures Reference Manual*.

Operational Characteristics

Conversation Management

Memory Systems Assistants maintain conversation state across multi-turn interactions, preserving context while managing memory limitations imposed by language model constraints. Memory systems support both short-term conversation continuity and long-term preference learning.

Context Window Optimization Efficient context management ensures optimal utilization of language model context windows while preserving conversation coherence. This includes intelligent summarization of conversation history and dynamic context prioritization.

Performance Optimization

Response Generation Language model integration supports various inference strategies including streaming responses for improved user experience and batch processing for high-throughput scenarios. Response generation incorporates retrieved knowledge while maintaining source attribution.

Retrieval Efficiency Knowledge base queries optimize for both accuracy and latency through configurable retrieval strategies. These include similarity thresholds, result ranking algorithms, and caching mechanisms for frequently accessed information.

Scalability Framework

Concurrent Operations Assistants support concurrent user interactions while maintaining individual conversation state isolation. Resource allocation scales dynamically based on demand patterns and organizational capacity constraints.

Resource Management Integration with Hybrid Manager infrastructure enables efficient resource utilization across assistant instances, including GPU allocation for model inference and storage optimization for conversation data.

Configuration Framework

Assistant Definition

Core Configuration Assistant creation requires specification of language model endpoints, knowledge base associations, and behavioral parameters. Configuration supports both template-based initialization and custom parameter specification.

Component Binding Knowledge bases, retrievers, rulesets, and tools connect to assistants through declarative configuration that maintains operational flexibility while ensuring consistent behavior across deployments.

Behavioral Configuration

Ruleset Application Behavioral guidelines define response characteristics including tone, formality, domain expertise, and operational boundaries. Rulesets support both static configuration and dynamic adaptation based on conversation context.

Response Parameters Language model parameters including temperature, token limits, and sampling strategies configure response generation characteristics. These parameters balance creativity with consistency based on application requirements.

Integration Configuration

Knowledge Base Connectivity Multiple knowledge bases connect to individual assistants through retriever configurations that specify search strategies, content filtering, and result ranking approaches. This architecture supports specialized knowledge domains and access control requirements.

Tool Authorization External system integration requires explicit authorization configuration that defines permitted operations, data access boundaries, and audit logging requirements. Tool configuration maintains security isolation while enabling necessary functionality.

Implementation Patterns

Enterprise Knowledge Systems

Departmental Expertise Assistants serve as subject matter expert interfaces for specific organizational domains, combining deep knowledge bases with appropriate behavioral characteristics for different user communities.

Cross-Functional Support Multi-domain assistants integrate knowledge from diverse organizational sources while maintaining appropriate access controls and information routing based on user permissions and query characteristics.

Customer-Facing Applications

Support Automation Customer service assistants combine product documentation, policy information, and troubleshooting knowledge to provide consistent support experiences while maintaining escalation pathways for complex issues.

Self-Service Portals Interactive knowledge access through conversational interfaces reduces support burden while providing comprehensive information discovery capabilities for customer self-service scenarios.

Internal Productivity Tools

Research Assistance Knowledge workers benefit from assistants that aggregate information across diverse internal sources, providing rapid access to organizational knowledge while maintaining appropriate attribution and source verification.

Decision Support Complex decision-making scenarios leverage assistants that combine quantitative data analysis with qualitative knowledge synthesis to support informed organizational choices.

Quality Assurance

Response Validation

Accuracy Verification Systematic evaluation procedures ensure assistant responses align with source knowledge while maintaining factual correctness. This includes both automated validation and human review processes for critical

applications.

Citation Integrity Source attribution accuracy supports organizational transparency and enables users to verify information independently. Citation systems maintain detailed provenance tracking from original sources through response generation.

Performance Monitoring

Conversation Analytics Comprehensive metrics track assistant effectiveness including user satisfaction, query resolution rates, and escalation patterns. Analytics support continuous improvement and optimization strategies.

System Performance Technical performance monitoring includes response latency, resource utilization, and system availability metrics that ensure consistent user experiences across varying load conditions.

Operational Considerations

Deployment Strategies

Development Workflows Assistant development follows systematic procedures including knowledge base validation, behavioral testing, and integration verification before production deployment. Development environments support iterative refinement and testing. See *Create an Assistant in Gen AI Builder* and SDK references.

Production Management Live assistant management includes performance monitoring, content updates, and behavioral adjustments based on operational feedback and changing organizational requirements.

Maintenance Procedures

Knowledge Base Updates Content synchronization ensures assistants access current organizational information while maintaining response consistency during transition periods. Update procedures support both incremental changes and comprehensive content refreshes.

Behavioral Refinement Ruleset modifications enable assistant behavior adaptation based on user feedback, organizational policy changes, and operational experience. Behavioral changes maintain consistency while supporting organizational evolution.

Security Framework

Access Control

User Authentication Assistant access integrates with organizational identity management systems while supporting both authenticated users and anonymous interactions based on deployment requirements and security policies.

Information Governance Knowledge base access respects organizational data classification and user permission frameworks, ensuring appropriate information exposure while maintaining comprehensive knowledge accessibility.

Audit Capabilities

Interaction Logging Detailed conversation logs support compliance requirements, operational analysis, and security monitoring while respecting privacy considerations and data retention policies.

Decision Traceability Response generation processes maintain comprehensive audit trails that enable verification of information sources, reasoning pathways, and decision factors for accountability and improvement purposes.

Integration Ecosystem

AI Factory Components

Assistants operate within the broader AI Factory ecosystem, leveraging capabilities from:

- **** Model serving and inference ****: Provides governed language model inference
- **** *AI Accelerator Pipelines* ****: Enables knowledge base content processing
- **** *Vector Engine (PGvector)* ****: Delivers semantic search infrastructure

Development Resources

Implementation Guides

- *Create an Assistant in Gen AI Builder* : Step-by-step assistant configuration
- *Knowledge Bases Reference Manual* : Comprehensive knowledge management
- *Tools Reference Manual* : External system integration patterns

Use Cases

- Enterprise knowledge assistants: KB + Retriever + Assistant + Tools (e.g., ticketing/CRM)
- Customer support copilots: policy/FAQ KBs + routing + guardrails via Rulesets
- Workflow bots: Structures + Tools for approvals, enrichment, reporting
- RAG for analytics: pgvector + Pipelines + Assistant for guided exploration

SDK Quick Links

- SDK overview: Gen AI Builder SDK
- Drivers: *Assistant/Prompt* , Conversation Memory
- Structures: *Agents* , *Tasks* , *Structures (workflows and building blocks)*
- Observability: *Structures/observability*

FAQ

- How do I wire an Assistant to multiple Knowledge Bases? - Create a *Create a Retriever in Gen AI Builder* targeting multiple KBs; assign it in *Create an Assistant in Gen AI Builder* . Use metadata filters per source.
- How do I let an Assistant call my internal APIs? - Package a *Create a Tool in Gen AI Builder* (or promote a *Create a Structure in Gen AI Builder*) with scoped credentials; attach it to the Assistant; enforce usage via *Create a Ruleset in Gen AI Builder* .
- How do I test and debug? - Use the test chat and *Threads Reference Manual* . Validate retrieval sets, check Ruleset revisions, and re-run with stored inputs. Export traces where needed.
- How do I choose models/drivers? - Start with general models; route task-specific prompts to cheaper/faster drivers; for private endpoints, integrate via Model serving and inference .

Assistants provide the foundational interface for intelligent applications within Gen AI Builder, orchestrating knowledge, behavior, and capabilities into coherent conversational experiences that operate within organizational governance frameworks.

Knowledge Bases Reference Manual

Knowledge Bases Reference Manual

Knowledge Bases provide semantic search infrastructure that transforms organizational content into queryable knowledge repositories, enabling AI assistants to access and utilize specific organizational knowledge through *retrieval* and *augmented generation* workflows. They serve as the operational interface between processed content and intelligent applications requiring contextual information.

Table of contents

- *Getting Started - AI Factory 101*
- *Practical Scenarios*
- *Architectural Purpose*
- *Technical Architecture*
- *Content Organization Framework*
- *Search Operation Types*
- *Implementation Patterns*
- *Operational Considerations*
- *Development Resources*

Getting Started

- Create a KB: *Create a Knowledge Base in Gen AI Builder*
- Prepare sources: *Configure Data Sources in Gen AI Builder* (Confluence, Google Drive, S3, Web Page, Data Lake, Custom)
- Configure storage: *Configure the Data Lake in Gen AI Builder* if using object storage ingestion
- Manage and update: *Manage Knowledge Bases in Gen AI Builder*
- Connect to apps: use *Retrievers Reference Manual* and *Assistants Reference Manual*

Practical Scenarios

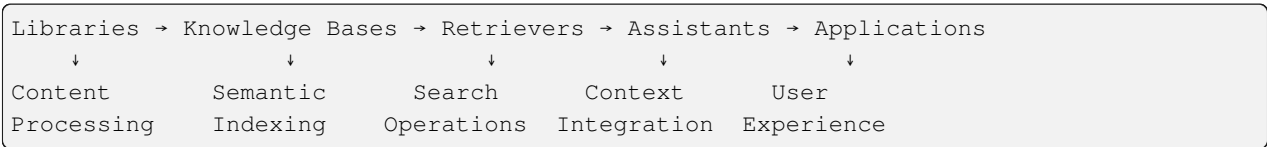
- Policy and procedures KB (enterprise internal): centralize HR/IT/Legal docs with strong citations. - Build steps: connect sources via *Configure Data Sources in Gen AI Builder* → *Create a Knowledge Base in Gen AI Builder* → validate retrieval with golden questions → attach via *Create a Retriever in Gen AI Builder* .
- Product documentation KB (support): ground agent answers on manuals and FAQs. - Build steps: ingest docs (S3/Drive/Confluence) → *Create a Knowledge Base in Gen AI Builder* → tune chunking/metadata → link to a support *Create an Assistant in Gen AI Builder* .
- Hybrid catalog KB (structured + unstructured): combine product attributes (structured) with descriptions/reviews (unstructured). - Build steps: prepare structured columns and unstructured text → create a Hybrid KB in *Create a Knowledge Base in Gen AI Builder* → expose filters in your *Create a Retriever in Gen AI Builder* .
- Analytics RAG KB: enable semantic exploration of reports and wiki knowledge. - Build steps: prepare embeddings via Pipelines and data preparation or Data Lake + loaders → *Create a Knowledge Base in Gen AI Builder* → test relevance and latency; integrate with an *Create an Assistant in Gen AI Builder* .

Architectural Purpose

Knowledge Bases function as the semantic search layer that converts static content collections into dynamic, queryable resources supporting AI applications with contextual understanding capabilities. They implement vector-based similarity search combined with metadata filtering to provide relevant information for language model context enhancement.

System Integration Framework

Gen AI Builder Pipeline Position Knowledge Bases occupy the critical position between content processing and application consumption within the AI Factory ecosystem:



Infrastructure Integration

- **Vector Engine:** Leverages PostgreSQL-based vector storage for high-performance semantic search operations
- **Libraries:** Consumes processed content with embedded vectors and enriched metadata
- **AI Accelerator Pipelines:** Utilizes embedding models for semantic indexing and content transformation
- **Model Serving:** Accesses language models for query understanding and response generation

Technical Architecture

Vector Storage Framework

Semantic Indexing Knowledge Bases implement sophisticated vector storage that captures semantic meaning through high-dimensional embeddings generated by specialized language models optimized for organizational content characteristics.

Storage Optimization Vector storage utilizes PostgreSQL’s native vector capabilities through the Vector Engine, providing enterprise-grade performance, scalability, and reliability while maintaining comprehensive query capabilities and operational management features.

Search Capabilities

Similarity Algorithms Knowledge Bases implement multiple similarity measurement approaches including cosine similarity, euclidean distance, and hybrid scoring mechanisms that combine semantic understanding with organizational priority factors.

Query Processing Pipeline Search operations follow systematic processing that optimizes result relevance while maintaining computational efficiency:

1. **Query Vectorization:** User queries transformed into semantic embeddings using organizational embedding models
2. **Similarity Computation:** Vector similarity calculations across indexed content collections
3. **Metadata Filtering:** Result refinement based on access controls, content classification, and organizational policies
4. **Relevance Ranking:** Multi-factor scoring incorporating semantic similarity, content recency, and source authority
5. **Context Assembly:** Result selection and formatting optimized for language model context integration

Content Organization Framework

Indexing Strategies

Content Segmentation Knowledge Bases implement intelligent content chunking that preserves semantic coherence while optimizing for retrieval granularity and language model context windows.

Hierarchical Organization Content structure preservation maintains document relationships, section hierarchies, and cross-reference connections enabling comprehensive information discovery and accurate source attribution.

Metadata Integration

Classification Systems Comprehensive metadata frameworks support organizational classification schemes including content types, access levels, subject classifications, and temporal relevance indicators.

Access Control Integration Metadata-based access control ensures appropriate information exposure aligned with organizational security policies and user permission frameworks.

Search Operation Types

Semantic Search

Vector Similarity Pure semantic search operations identify content based on conceptual similarity regardless of keyword matching, enabling discovery of relevant information through meaning rather than literal text matching.

Contextual Understanding Advanced semantic search incorporates conversation context, user intent, and organizational domain knowledge for enhanced relevance and accuracy in information discovery.

Hybrid Search Operations

Combined Approaches Hybrid search strategies integrate vector similarity with traditional keyword matching, metadata filtering, and organizational priority factors for optimal balance between semantic understanding and precise information retrieval.

Performance Optimization Intelligent query routing selects optimal search strategies based on query characteristics, content types, and performance requirements ensuring consistent response times while maximizing result relevance.

Filtered Search

Metadata-Based Filtering Advanced filtering capabilities enable precise content selection based on organizational classification, access permissions, content recency, and source credibility indicators.

Dynamic Filtering Context-aware filtering adjusts search scope based on user roles, conversation context, and organizational policies ensuring appropriate information exposure while maintaining comprehensive coverage.

Performance Characteristics

Scalability Framework

Horizontal Scaling Knowledge Base architecture supports horizontal scaling across multiple PostgreSQL instances while maintaining search consistency and performance characteristics through intelligent load distribution.

Query Optimization Advanced query planning optimizes search operations for response time consistency across growing content volumes through intelligent indexing strategies and caching mechanisms.

Resource Management

Memory Optimization Vector storage optimization balances comprehensive content coverage with memory efficiency through intelligent indexing strategies and storage compression techniques.

Computational Efficiency Search operations optimize computational resource utilization through intelligent query planning, result caching, and parallel processing capabilities.

Quality Assurance

Content Quality Management

Ingestion Validation Systematic content validation during knowledge base creation ensures information accuracy, completeness, and appropriate formatting for optimal search effectiveness.

Ongoing Quality Assessment Continuous monitoring of search effectiveness and content quality supports systematic improvement through performance analytics and user feedback integration.

Search Effectiveness

Relevance Monitoring Comprehensive assessment of search result relevance supports continuous optimization through systematic analysis of user interactions, feedback patterns, and task completion rates.

Performance Tracking Detailed performance monitoring tracks search response times, result accuracy, and resource utilization patterns supporting capacity planning and optimization decisions.

Implementation Patterns

Enterprise Knowledge Systems

Organizations implement comprehensive knowledge bases that unify diverse content sources while maintaining appropriate access controls and organizational governance requirements.

Unified Access Architecture

- Cross-departmental content integration through centralized knowledge base infrastructure
- Role-based access control ensuring appropriate information exposure
- Comprehensive audit trails supporting compliance and operational oversight
- Performance optimization supporting concurrent access and organizational scalability

Specialized Domain Applications

Domain-specific implementations optimize knowledge bases for particular industries or use cases requiring specialized content organization and search characteristics.

Domain Optimization Features

- Industry-specific embedding models optimized for domain terminology and concepts
- Specialized metadata schemas supporting domain-specific classification and organization
- Regulatory compliance integration ensuring industry-specific requirement adherence
- Performance optimization for domain-specific query patterns and content characteristics

Multi-Modal Knowledge Integration

Advanced implementations coordinate knowledge across diverse content types including documents, structured data, and multimedia resources while maintaining unified search experiences.

Cross-Modal Capabilities

- Content type optimization ensuring appropriate handling across diverse media formats
- Unified search interfaces providing consistent access across different information types
- Performance optimization supporting diverse content processing and retrieval requirements
- Quality assurance procedures ensuring consistent results across content modalities

Operational Considerations

Deployment Management

Configuration Management Knowledge Base deployment requires systematic configuration management ensuring consistent search behavior across development, staging, and production environments while supporting organizational governance requirements.

Update Procedures Content updates require coordinated index maintenance ensuring search effectiveness remains optimal while minimizing service disruption during content refresh operations.

Maintenance Framework

Index Optimization Regular maintenance procedures ensure optimal search performance through index rebuilding, vector optimization, and storage consolidation supporting system efficiency and user experience.

Content Synchronization Systematic procedures coordinate content updates from Libraries with knowledge base refresh operations ensuring information currency while maintaining search performance characteristics.

Development Resources

Configuration Procedures

Knowledge Base Setup

- *Create a Knowledge Base in Gen AI Builder* : Comprehensive knowledge base creation and configuration procedures
- *Manage Knowledge Bases in Gen AI Builder* : Ongoing management and optimization techniques
- *Best practices for Hybrid Knowledge Bases* : Advanced configuration for optimal search effectiveness

Integration Documentation

System Connectivity

- *LibrariesΩ Manual* : Content source configuration and processing optimization
- *Retrievers Reference Manual* : Search interface configuration and optimization
- *Assistants Reference Manual* : Knowledge base utilization within conversational systems

Technical Reference

- Vector Store Drivers : Technical implementation details
- Embedding Integration : Semantic indexing configuration

- RAG Engine Reference : Advanced retrieval-augmented generation implementation
-

Knowledge Bases transform static organizational content into dynamic, intelligent resources that enable AI applications to provide contextually relevant, accurate responses grounded in specific organizational knowledge while maintaining appropriate access controls and performance characteristics.

Configure Data Sources in Gen AI Builder

Who is this for

Platform users building Knowledge Bases and AI applications using Gen AI Builder. This includes developers, AI engineers, content managers, and system integrators.

If you are responsible for bringing external or internal content into your AI workflows—whether from documents, web pages, cloud storage, or proprietary systems—this set of guides will show you how.

What you will accomplish

You will learn how to configure and manage different types of Data Sources in Gen AI Builder. These sources provide the raw content that populates Libraries and Knowledge Bases for your AI applications.

Why use Data Sources

- The quality and relevance of your AI applications depend heavily on the content you ingest.
- Gen AI Builder allows you to connect to a variety of content sources and manage the ingestion pipeline securely and efficiently.
- Using Data Sources ensures your Knowledge Bases stay fresh, accurate, and aligned with your organization's needs.

About Data Sources in Gen AI Builder

Data Sources are the first step in bringing content into the system. The content from Data Sources flows into **Libraries**, and then into **Knowledge Bases**, where it is indexed and made available for AI workloads.

Common types of Data Sources include:

- Web pages
- Data Lakes
- Amazon S3 buckets
- Google Drive folders
- Atlassian Confluence spaces
- Custom systems via PG.AI Structures

Key features:

- Secure connection to data
- Scheduled refresh to keep content up to date
- Optional transformation with PG.AI Structures
- Integration with downstream Libraries and Knowledge Bases

For foundational concepts, see:

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Structures Reference Manual*
- *Embeddings explained*
- *Retrieval Augmented Generation (RAG)*

For guidance on using this content in AI applications:

- Knowledge Bases overview
- *Create a Knowledge Base in Gen AI Builder*

To see how Data Sources are used within **Hybrid Manager deployments**, visit:

- [Hybrid Manager: Using Gen AI Builder](#)

How-to Guides

Use the following guides to configure each type of Data Source:

- *Configure a Web Page data source*
- *Configure a Data Lake data source*
- *Configure an Amazon S3 data source*
- *Configure a Google Drive data source*
- *Configure an Atlassian Confluence data source*
- *Configure a Custom data source*

Each guide provides:

- Who it is for
- What you will accomplish
- Why to use this source
- Detailed configuration steps
- Key considerations and best practices
- Troubleshooting tips
- Example scenarios

Next steps

Once you have configured Data Sources, proceed to:

- Configure Libraries to organize your content.
- Build Knowledge Bases to enable semantic search and RAG applications.
- Connect your Knowledge Bases to AI agents and workflows using Gen AI Builder and Structures.

For more advanced integrations, see:

- *Create a Knowledge Base in Gen AI Builder*
- *Structures Reference Manual*

SDK reference

- Data Loaders
- File Manager Drivers
- Web Scraper Drivers

By thoughtfully configuring Data Sources, you will lay a strong foundation for high-quality, context-aware AI applications in your organization.

Retrievers Reference Manual

Retrievers Reference Manual

Retrievers provide specialized information retrieval capabilities that enable AI assistants to access and utilize organizational knowledge through precise, configurable search operations. They serve as the critical interface between user queries and knowledge base content, implementing the retrieval component of retrieval-augmented generation workflows.

Architectural Function

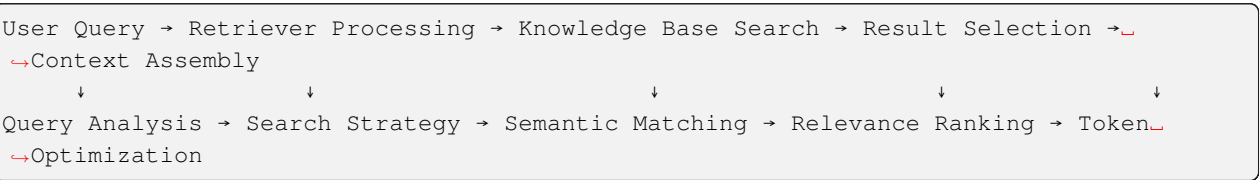
Retrievers operate as specialized Griptape Structures that orchestrate semantic search operations across knowledge bases while applying organizational constraints and optimization strategies. They transform user queries into structured search operations that identify relevant content for language model context integration.

System Integration Framework

Gen AI Builder Integration Retrievers integrate seamlessly with core Gen AI Builder components to enable comprehensive knowledge-driven applications:

- **Knowledge Bases:** Execute semantic search operations across indexed content collections
- **Assistants:** Provide contextual information for informed response generation
- **Libraries:** Access processed content through optimized retrieval interfaces
- **Vector Engine:** Leverage high-performance semantic search capabilities within PostgreSQL infrastructure

Retrieval Pipeline Architecture



This pipeline ensures optimal content discovery while managing computational resources and maintaining response quality standards.

Technical Architecture

Retrieval Strategy Framework

Search Operation Types Retrievers implement multiple search strategies optimized for different content types and organizational requirements:

Strategy Type	Primary Function	Optimization Focus
Semantic Search	Vector similarity matching	Content relevance and contextual understanding
Hybrid Search	Combined vector and keyword approaches	Precision balance between semantic and exact matching
Filtered Search	Metadata-based content selection	Access control and content classification adherence
Multi-Source Search	Cross-knowledge base retrieval	Comprehensive information coverage

Query Processing Pipeline Retriever operations follow systematic query processing that optimizes search effectiveness while maintaining performance requirements:

- 1. **Query Analysis:** User input parsing and intent classification
- 2. **Search Strategy Selection:** Optimal approach determination based on query characteristics
- 3. **Knowledge Base Targeting:** Appropriate content source identification and prioritization
- 4. **Semantic Matching:** Vector similarity computation and relevance scoring
- 5. **Result Assembly:** Content selection and context window optimization

Content Selection Framework

Relevance Optimization Retrievers implement sophisticated relevance scoring that combines semantic similarity with organizational priority factors including content recency, source authority, and user access permissions.

Context Management Token limit management ensures optimal language model context utilization through intelligent content selection, summarization, and prioritization strategies that maintain information completeness within computational constraints.

Configuration Architecture

Knowledge Base Targeting

Source Selection Retriever configuration defines specific knowledge bases for search operations, enabling precise control over information scope and ensuring appropriate content access based on organizational requirements.

Multi-Source Coordination Advanced retrievers coordinate searches across multiple knowledge bases while managing result integration, deduplication, and relevance ranking across diverse content sources.

Performance Parameters

Token Limits Maximum token configuration controls retrieved content volume, balancing comprehensive information provision with language model context window limitations and processing efficiency requirements.

Search Depth Result quantity parameters define the breadth of content consideration during retrieval operations, optimizing between comprehensive coverage and computational resource utilization.

Metadata Integration

Content Classification Metadata utilization enables sophisticated content filtering based on organizational classification schemes, access control requirements, and content quality indicators.

Contextual Enhancement Advanced metadata integration supports query enhancement, result ranking optimization, and content attribution requirements for organizational transparency and audit capabilities.

Operational Characteristics

Search Performance

Response Optimization Retrievers optimize search operations for consistent response times while maintaining result quality through intelligent caching, index optimization, and query planning strategies.

Scalability Framework Retrieval operations scale efficiently across growing knowledge base collections through distributed search capabilities and intelligent resource allocation strategies.

Quality Assurance

Result Validation Systematic result quality assessment ensures retrieved content meets relevance thresholds and organizational quality standards through automated validation and continuous monitoring procedures.

Relevance Monitoring Ongoing assessment of retrieval effectiveness supports continuous optimization through performance analytics, user feedback integration, and systematic improvement procedures.

Implementation Patterns

Enterprise Knowledge Access

Organizations implement comprehensive retrieval systems that provide unified access to diverse knowledge sources while maintaining appropriate access controls and organizational governance requirements.

Unified Search Architecture

- Cross-departmental knowledge integration through centralized retrieval capabilities
- Role-based content access ensuring appropriate information exposure
- Comprehensive audit trails supporting compliance and operational oversight
- Performance optimization supporting concurrent user access and system scalability

Specialized Domain Applications

Domain-specific implementations optimize retrieval operations for particular industries or use cases requiring specialized knowledge organization and access patterns.

Domain Optimization Features

- Industry-specific relevance scoring algorithms
- Specialized metadata utilization for domain-specific content organization
- Regulatory compliance integration supporting industry-specific requirements
- Performance optimization for domain-specific query patterns and content characteristics

Multi-Modal Information Systems

Advanced implementations coordinate retrieval across diverse content types including documents, structured data, and multimedia resources while maintaining unified search experiences.

Cross-Modal Integration

- Content type optimization ensuring appropriate handling across diverse media formats
- Unified relevance scoring across different information types and structures
- Performance optimization supporting diverse content processing requirements

- Quality assurance procedures ensuring consistent results across content modalities

Quality Framework

Retrieval Effectiveness

Relevance Assessment Systematic evaluation procedures verify retrieval accuracy and effectiveness through standardized metrics including precision, recall, and user satisfaction measurements.

Performance Monitoring Comprehensive monitoring tracks retrieval performance characteristics including response times, result quality, and resource utilization patterns supporting optimization and capacity planning decisions.

Continuous Improvement

Optimization Procedures Regular assessment and optimization of retrieval parameters support continuous improvement through systematic analysis of performance characteristics and user feedback integration.

Feedback Integration User interaction analysis and feedback mechanisms inform retrieval optimization enabling systematic enhancement of search effectiveness and user experience quality.

Operational Considerations

Deployment Management

Configuration Management Retriever deployment requires systematic configuration management ensuring consistent behavior across development, staging, and production environments while supporting organizational governance requirements.

Version Control Systematic versioning procedures support reliable deployment management and rollback capabilities when operational issues arise or optimization requirements change.

Maintenance Procedures

Performance Optimization Regular maintenance procedures ensure optimal retrieval performance through index optimization, cache management, and query pattern analysis supporting system efficiency and user experience.

Content Synchronization Knowledge base updates require coordinated retriever optimization ensuring search effectiveness remains optimal as content evolves and organizational requirements change.

Integration Resources

Configuration Procedures

Retriever Setup

- *Create a Retriever in Gen AI Builder* : Comprehensive retriever configuration and deployment procedures
- *Retrievers Reference Manual* : Advanced configuration techniques for optimal search effectiveness

System Integration

Component Connectivity

- *Assistants Reference Manual* : Retriever utilization within conversational systems
- *LibrariesΩ Manual* : Content source configuration and optimization
- *Structures Reference Manual* : Advanced retrieval workflow development

Infrastructure Integration

- Vector Engine Integration : High-performance semantic search capabilities
- Hybrid Manager Deployment : Infrastructure management and resource optimization

SDK reference

- RAG Engines
 - Embedding Drivers
 - Rerank Drivers
 - Vector Store Drivers
-

Retrievers provide essential information access capabilities that transform static knowledge repositories into dynamic, queryable resources supporting intelligent applications with comprehensive organizational knowledge integration and semantic understanding.

Rulesets Manual

Rulesets provide behavioral governance for AI assistants through natural language instructions that define response patterns, communication standards, and operational constraints. They establish explicit behavioral controls that complement knowledge retrieval systems to ensure consistent, compliant interactions across organizational AI applications.

Architectural Function

Rulesets serve as the behavioral control layer within Gen AI Builder, defining how assistants interpret and respond to user interactions while maintaining organizational standards and compliance requirements. They operate at the prompt level, influencing language model behavior before response generation occurs.

System Integration Context

Gen AI Builder Components

- **Assistants:** Apply rulesets during conversation processing to ensure behavioral consistency
- **Knowledge Bases:** Work alongside retrieval systems to combine factual accuracy with appropriate behavior
- **Structures:** Incorporate ruleset guidance within complex workflow operations
- **Tools:** Apply behavioral constraints during external system interactions

Behavioral Control Layer Rulesets function as a middleware component that processes user inputs and assistant responses according to defined organizational guidelines. This control layer ensures consistent application of behavioral standards regardless of conversation complexity or topic variation.

Core Architecture

Ruleset Components

Structural Elements

- **Name:** Unique identifier for organizational reference and management
- **Description:** Documentation of ruleset purpose and intended application
- **Rules Collection:** Individual behavioral instructions written in natural language

- **Alias:** Optional identifier for programmatic access and versioning
- **Metadata:** Additional configuration parameters for advanced behavioral control

Rule Definition Framework Individual rules within rulesets consist of:

- **Rule Name:** Descriptive identifier for specific behavioral instruction
- **Rule Text:** Natural language instruction defining expected behavior
- **Priority Level:** Implicit ordering for rule application when conflicts arise
- **Context Scope:** Application boundaries for rule enforcement

Behavioral Application Process

Runtime Integration Rulesets integrate into assistant processing through systematic application during conversation flow:

1. **Input Processing:** User messages evaluated against ruleset constraints
2. **Context Establishment:** Relevant rules activated based on conversation context
3. **Response Generation:** Language model guided by active ruleset instructions
4. **Output Validation:** Generated responses verified against behavioral requirements
5. **Consistency Enforcement:** Ongoing behavioral monitoring throughout conversation

Rule Precedence and Conflict Resolution When multiple rules apply to similar situations, the system applies precedence based on rule specificity, organizational priority levels, and contextual relevance. This approach ensures consistent behavior while maintaining flexibility for complex scenarios.

Behavioral Categories

Communication Standards

Professional Communication Rules defining appropriate tone, language formality, and interaction patterns for different organizational contexts and user relationships.

Brand Alignment Behavioral guidelines ensuring assistant responses reflect organizational voice, values, and communication standards across all interactions.

Cultural Sensitivity Instructions for appropriate communication across diverse user populations while respecting cultural differences and organizational inclusion policies.

Compliance and Governance

Regulatory Compliance Rules enforcing industry-specific regulatory requirements including data privacy, financial services regulations, and healthcare compliance standards.

Information Governance Behavioral constraints defining appropriate information sharing, source attribution requirements, and confidentiality protection measures.

Escalation Protocols Instructions defining when and how assistants should escalate conversations to human operators based on complexity, sensitivity, or regulatory requirements.

Operational Boundaries

Capability Limitations Clear definitions of assistant capabilities and limitations, ensuring appropriate expectation setting and service boundary communication.

Error Handling Behavioral guidelines for managing system failures, information gaps, and unexpected situations while maintaining user confidence and service quality.

Security Protocols Rules defining appropriate handling of security-related inquiries, authentication procedures, and sensitive information protection measures.

Implementation Patterns

Customer Service Applications

Customer-facing assistants implement comprehensive behavioral frameworks that ensure consistent service delivery while maintaining organizational brand standards and regulatory compliance requirements.

Service Excellence Rules

- Professional tone maintenance across all interaction types
- Proactive assistance offering and problem resolution approaches
- Clear communication of capabilities and limitations
- Appropriate escalation trigger recognition and execution

Internal Knowledge Systems

Enterprise knowledge assistants utilize behavioral rulesets that balance information accessibility with organizational security requirements while maintaining productivity and user experience standards.

Knowledge Governance Rules

- Source attribution requirements for all factual claims
- Appropriate information classification and access control respect
- Clear uncertainty communication when information confidence is low
- Documentation of reasoning processes for audit and compliance purposes

Specialized Domain Applications

Industry-specific assistants implement domain expertise behavioral patterns that combine technical accuracy with appropriate professional communication standards and regulatory compliance requirements.

Domain Expertise Rules

- Technical terminology usage appropriate to user expertise levels
- Accuracy disclaimers for complex technical or regulatory information
- Professional liability protection through appropriate limitation statements
- Industry best practice integration in guidance and recommendations

Configuration Framework

Ruleset Development

Rule Creation Process Effective rulesets require systematic development that considers organizational requirements, user expectations, and operational constraints while maintaining clarity and enforceability.

Development Considerations

- Clear, specific instruction language avoiding ambiguous or conflicting guidance
- Appropriate scope definition preventing over-broad or under-specified behavioral constraints
- Organizational alignment ensuring rules reflect actual business requirements and values
- Testability considerations enabling validation of rule effectiveness and compliance

Rule Management

Version Control Rulesets require systematic version management enabling controlled updates, rollback capabilities, and historical tracking of behavioral modifications.

Change Management Rule modifications follow established change control procedures including testing, approval workflows, and deployment coordination to minimize operational disruption.

Quality Assurance

Behavioral Validation

Consistency Testing Systematic evaluation procedures verify ruleset effectiveness across diverse conversation scenarios and ensure consistent behavioral application under varying conditions.

Compliance Verification Regular assessment of rule adherence and effectiveness in meeting organizational compliance requirements and regulatory standards.

Performance Assessment

User Experience Impact Monitoring behavioral rule impact on user satisfaction, task completion rates, and overall service effectiveness to inform optimization decisions.

Operational Efficiency Assessment of ruleset impact on system performance, response times, and resource utilization to balance behavioral control with operational requirements.

Operational Considerations

Rule Maintenance

Ongoing Review Regular evaluation of rule effectiveness, relevance, and organizational alignment to ensure continued applicability and value.

Performance Monitoring Systematic tracking of rule application effectiveness, user feedback, and compliance outcomes to support continuous improvement efforts.

Organizational Integration

Stakeholder Alignment Coordination with legal, compliance, and business teams to ensure ruleset alignment with organizational policies and regulatory requirements.

Training Integration Alignment of assistant behavioral standards with human agent training and organizational service standards to maintain consistency across service channels.

Development Resources

Creation Guidelines

Implementation Procedures

- *Create a Ruleset in Gen AI Builder* : Comprehensive ruleset development and deployment procedures
- *How-To View and Manage Threads in Gen AI Builder in Hybrid Manager* : Validation methods for ruleset effectiveness assessment

Integration Documentation

Component Integration

- *Assistants Reference Manual* : Ruleset assignment and behavioral integration procedures
- *Structures Reference Manual* : Behavioral governance within complex workflows

AI Factory Infrastructure

- Gen AI Builder Overview : Platform architecture and component relationships
- *Hybrid Manager Integration* : Infrastructure management and deployment procedures

SDK reference

- *Rulesets in Structures*
 - Ruleset Drivers
-

Rulesets provide essential behavioral governance that transforms AI systems from technically capable tools into organizationally aligned applications that consistently reflect institutional values, compliance requirements, and service standards across all user interactions.

Structures Reference Manual

Structures provide advanced workflow orchestration capabilities that enable multi-step AI processing pipelines, external system integration, and custom business logic execution within Gen AI Builder. They transform simple conversational AI into sophisticated enterprise applications capable of complex reasoning, data transformation, and system automation.

Architectural Purpose

Structures serve as the orchestration engine for complex AI workflows that require multiple processing steps, external service integration, and custom logic implementation. They extend beyond basic retrieval-augmented generation to enable enterprise-grade applications with sophisticated reasoning capabilities and operational integration requirements.

System Integration Framework

Gen AI Builder Components

- **Assistants**: Invoke structures through tool interfaces for complex task execution
- **Data Sources**: Utilize structures for content transformation and enrichment pipelines
- **Knowledge Bases**: Leverage structures for custom retrieval logic and content processing
- **Tools**: Implement structures as sophisticated tools with multi-step capabilities

External System Connectivity

- **API Integration:** RESTful and GraphQL service orchestration across multiple endpoints
- **Database Operations:** Complex data processing workflows spanning multiple data sources
- **Business Systems:** Enterprise resource planning, customer relationship management, and financial system integration
- **Workflow Automation:** Process orchestration across diverse organizational systems

Technical Architecture

Structure Implementation Framework

Structures implement sophisticated workflow patterns using the Griptape framework, providing standardized interfaces for complex multi-step operations while maintaining compatibility with Gen AI Builder orchestration requirements.

Implementation Foundation Structures are developed using the Griptape Python framework with integration patterns optimized for Gen AI Builder deployment. The architecture supports various workflow types including sequential pipelines, parallel processing, and conditional branching based on business logic requirements.

Core Structure Categories

Structure Type	Primary Function	Use Cases
Agent Structures	Autonomous decision-making and task execution	Complex problem-solving, multi-step reasoning
Pipeline Structures	Sequential data transformation workflows	Content processing, data enrichment
Workflow Structures	Conditional branching and parallel processing	Business process automation, approval workflows

Execution Architecture

Structure Invocation Patterns

1. **Direct Execution:** Manual structure invocation through console interfaces
2. **API Integration:** Programmatic execution through RESTful API endpoints
3. **Assistant Integration:** Structure invocation through assistant tool interfaces
4. **Pipeline Integration:** Automated execution within data source processing workflows
5. **Event-Driven Execution:** Triggered execution based on system events or schedules

Processing Coordination Structures coordinate complex operations through systematic orchestration of multiple components including language models, external APIs, data sources, and business logic processors. This coordination maintains state consistency while enabling sophisticated workflow patterns.

Structure Categories

Agent-Based Structures

Autonomous Reasoning Agents Complex decision-making structures that combine multiple information sources with sophisticated reasoning capabilities to solve multi-faceted problems requiring contextual understanding and strategic thinking.

Capabilities:

- Multi-source information synthesis and analysis
- Dynamic tool selection based on problem characteristics
- Iterative problem-solving with feedback incorporation
- Context-aware decision making across extended interactions

Data Processing Pipelines

Content Transformation Structures Systematic data processing workflows that transform raw content into enriched, structured information suitable for knowledge base integration and semantic search operations.

Processing Stages:

- **Content Extraction:** Format-specific parsing and content identification
- **Enrichment:** Metadata addition, classification, and contextual annotation
- **Transformation:** Format conversion, structure optimization, and quality enhancement
- **Validation:** Content accuracy verification and completeness assessment

Integration Workflows

External System Orchestration Complex integration structures that coordinate operations across multiple external systems while maintaining data consistency, error handling, and transaction integrity.

Integration Patterns:

- **API Orchestration:** Multi-service coordination with dependency management
- **Data Synchronization:** Cross-system data consistency maintenance
- **Process Automation:** End-to-end business process implementation
- **Error Recovery:** Sophisticated error handling and retry mechanisms

Implementation Patterns

Enterprise Data Processing

Organizations implement comprehensive data processing structures that transform diverse information sources into unified, searchable knowledge repositories while maintaining data quality and governance standards.

Architecture Components

- **Source Adapters:** System-specific data extraction and normalization
- **Processing Engines:** Content transformation and enrichment workflows
- **Quality Assurance:** Validation and verification procedures
- **Output Optimization:** Format and structure optimization for downstream consumption

Business Process Automation

Structures enable sophisticated business process automation that combines human decision-making with system automation while maintaining appropriate oversight and control mechanisms.

Automation Capabilities

- **Approval Workflows:** Multi-stakeholder decision processes with routing logic

- **Compliance Verification:** Automated policy adherence checking and validation
- **Exception Handling:** Intelligent exception identification and resolution workflows
- **Audit Trail Generation:** Comprehensive activity logging and compliance documentation

Custom Retrieval Systems

Advanced retrieval structures implement sophisticated search strategies that combine semantic similarity with business logic, contextual filtering, and dynamic result optimization based on user characteristics and organizational requirements.

Retrieval Enhancement Features

- **Hybrid Search:** Vector similarity combined with rule-based filtering
- **Contextual Adaptation:** Search strategy adjustment based on user context and permissions
- **Result Optimization:** Dynamic ranking and filtering based on relevance criteria
- **Performance Optimization:** Caching and indexing strategies for improved response times

Configuration Framework

Structure Definition

Structures require comprehensive configuration that defines processing logic, external system connectivity, and operational parameters while maintaining security boundaries and performance optimization.

Configuration Components

Deployment Configuration

Structure deployment requires systematic configuration management that ensures consistent behavior across development, staging, and production environments while maintaining appropriate security controls.

Deployment Parameters

- **Resource Allocation:** CPU, memory, and processing time limitations
- **Security Configuration:** Authentication, authorization, and data access controls
- **Performance Optimization:** Caching strategies, parallel processing, and resource utilization
- **Monitoring Integration:** Logging, metrics collection, and performance tracking

Quality Assurance

Performance Monitoring

Operational Metrics Structures implement comprehensive monitoring that tracks performance characteristics, resource utilization, and operational effectiveness across diverse execution scenarios.

Key Performance Indicators

- **Execution Latency:** Processing time measurements across different workflow complexity levels
- **Success Rates:** Completion percentages for various structure types and configurations
- **Resource Utilization:** CPU, memory, and network resource consumption patterns
- **Error Patterns:** Systematic analysis of failure modes and recovery effectiveness

Reliability Framework

Error Handling and Recovery Sophisticated error handling mechanisms ensure graceful degradation and recovery from various failure scenarios while maintaining data integrity and operational continuity.

Reliability Features

- **Retry Mechanisms:** Intelligent retry strategies with exponential backoff and circuit breaker patterns
- **Fallback Procedures:** Alternative processing paths for critical operations
- **State Management:** Transaction consistency and rollback capabilities
- **Monitoring Integration:** Real-time error detection and alerting systems

Operational Considerations

Deployment Management

Structure Lifecycle

- **Version Control:** Systematic versioning with backward compatibility considerations
- **Deployment Automation:** Consistent deployment procedures across environments
- **Rollback Capabilities:** Automated rollback mechanisms for problematic deployments
- **Documentation Maintenance:** Comprehensive documentation aligned with structure evolution

Maintenance Procedures

Ongoing Operations

- **Performance Optimization:** Continuous performance tuning based on usage patterns and feedback
- **Security Updates:** Regular security assessment and vulnerability remediation
- **Dependency Management:** External system connectivity monitoring and maintenance
- **Capacity Planning:** Resource allocation optimization based on growth projections

Development Framework

Implementation Guidelines

Structure development follows systematic procedures that ensure consistency, maintainability, and operational reliability across diverse organizational requirements.

Development Best Practices

- **Modular Design:** Component separation enabling independent testing and maintenance
- **Error Handling:** Comprehensive error management with appropriate user feedback
- **Security Integration:** Built-in security controls aligned with organizational policies
- **Performance Optimization:** Efficient resource utilization and scalability considerations

Testing Strategies

Validation Framework

- **Unit Testing:** Individual component functionality verification

- **Integration Testing:** End-to-end workflow validation across system boundaries
- **Performance Testing:** Load and stress testing under expected operational conditions
- **Security Testing:** Vulnerability assessment and penetration testing procedures

Implementation Resources

Development Documentation

Creation Procedures

- *Create a Structure in Gen AI Builder* : Comprehensive structure development and deployment procedures
- *Structure Configuration* : Detailed configuration options and best practices
- *Structure Execution* : Testing and validation procedures for deployed structures

Integration Resources

Gen AI Builder Components

- *Assistants Reference Manual* : Structure utilization within conversational interfaces
- *Tools Reference Manual* : Structure implementation as sophisticated tools
- *LibrariesΩ Manual* : Structure utilization in content processing workflows

AI Factory Infrastructure

- *Model Serving integration* : Language model connectivity for structure operations
- Hybrid Manager Deployment : Infrastructure integration and resource management

SDK reference

- *Structures Reference Manual*
 - *Tasks* , Pipelines and data preparation , *Structures (workflows and building blocks)*
 - *Tools Reference Manual*
-

Structures enable organizations to implement sophisticated AI-powered workflows that combine reasoning, integration, and automation capabilities within governed, scalable infrastructure environments that support enterprise operational requirements.

Tools Reference Manual

Tools Reference Manual

Tools provide extensible action capabilities that transform passive AI assistants into active systems capable of interacting with external services, performing computations, and executing custom business logic. They serve as the bridge between conversational AI interfaces and operational systems within organizational infrastructure.

Architectural Purpose

Tools extend language model capabilities beyond text generation by providing structured interfaces for external system integration. They enable assistants to perform concrete actions including data retrieval, system operations, and workflow automation while maintaining security boundaries and operational governance.

System Integration Framework

Gen AI Builder Components

- **Assistants:** Invoke tools through language model decision-making processes
- **Structures:** Incorporate tools as workflow components within pipeline operations
- **Rulesets:** Define tool usage policies and operational constraints
- **Memory Systems:** Maintain context for multi-step tool execution sequences

External System Connectivity

- **APIs:** RESTful and GraphQL service integration for data exchange
- **Databases:** Direct database connectivity for operational data access
- **Services:** Internal system integration through authenticated connections
- **Workflows:** Business process automation through multi-tool orchestration

Technical Architecture

Tool Implementation Framework

Tools implement standardized interfaces that enable consistent integration across different assistant types and operational contexts. The framework supports both simple single-purpose tools and complex multi-step workflow automation.

Implementation Reference Tools are developed using the pgai-ai-builder SDK framework available at [EnterpriseDB/pgai-ai-builder](#). This SDK provides the foundational classes and utilities for tool development.

Execution Architecture

Tool Invocation Flow

1. **Decision Phase:** Language models determine tool necessity based on conversation context
2. **Parameter Extraction:** Relevant information extraction from user inputs and conversation history
3. **Security Validation:** Permission verification and rate limit enforcement
4. **Execution Phase:** Tool-specific logic execution with error handling and timeout management
5. **Result Integration:** Output formatting and incorporation into conversation flow
6. **Context Updates:** Memory system updates reflecting tool execution results

Tool Categories

System Integration Tools

API Connectors External service integration tools that provide standardized interfaces for diverse API types including REST, GraphQL, and service-specific protocols.

Common Implementations:

- **CRM Integration:** Customer relationship management system connectivity
- **ERP Systems:** Enterprise resource planning data access
- **Payment Processors:** Financial transaction processing capabilities
- **Communication Services:** Email, SMS, and notification system integration

Database Access Tools Direct database connectivity tools that enable structured data queries while maintaining security boundaries and performance optimization.

Supported Database Types:

- **PostgreSQL:** Native integration with Vector Engine capabilities
- **Cloud Databases:** AWS RDS, Google Cloud SQL, Azure Database
- **Data Warehouses:** Snowflake, BigQuery, Redshift
- **NoSQL Systems:** MongoDB, Redis, Elasticsearch

Computational Tools

Mathematical Operations Specialized computational tools that perform calculations beyond language model capabilities, including statistical analysis, financial modeling, and scientific computations.

Business Logic Processors Custom business rule implementation tools that encode organizational logic including pricing calculations, eligibility determinations, and compliance validations.

Data Transformation Tools Content processing tools that manipulate data formats, perform aggregations, and execute complex transformations required for business operations.

Workflow Automation Tools

Process Orchestration Multi-step workflow coordination tools that manage complex business processes including approval workflows, data synchronization, and system integration sequences.

Notification Systems Communication coordination tools that manage alert distribution, status updates, and stakeholder communication across multiple channels and systems.

Implementation Patterns

Enterprise System Integration

Organizations implement comprehensive tool suites that connect AI assistants with existing operational systems while maintaining security boundaries and performance requirements.

Security Framework

- **Authentication Integration:** Organizational identity system connectivity
- **Permission Boundaries:** Role-based access control for tool operations
- **Audit Logging:** Comprehensive operation tracking for compliance requirements
- **Rate Limiting:** Resource protection through configurable usage constraints

Customer Service Automation

Customer-facing applications leverage specialized tools that provide real-time access to customer information, order status, and support system integration.

Typical Tool Suite

- **Account Lookup:** Customer information retrieval from CRM systems
- **Order Management:** Order status queries and modification capabilities
- **Payment Processing:** Secure financial transaction handling
- **Ticket Management:** Support case creation and status updates

Internal Productivity Enhancement

Knowledge worker applications utilize tools that integrate with productivity systems including document management, project tracking, and communication platforms.

Productivity Integration Tools

- **Document Retrieval:** Enterprise content management system access
- **Calendar Integration:** Schedule management and meeting coordination
- **Project Management:** Task creation and status tracking across project systems
- **Analytics Access:** Business intelligence and reporting system integration

Configuration Management

Tool Configuration Schema

Tools require systematic configuration that defines operational parameters, security boundaries, and integration requirements.

Environment Management

Tool deployment requires systematic configuration management that balances operational flexibility with security requirements.

Quality Assurance

Performance Monitoring

Operational Metrics

- **Execution Latency:** Tool operation completion time measurements
- **Success Rates:** Operational success percentages across different tool types
- **Error Patterns:** Systematic analysis of failure modes and error conditions
- **Resource Utilization:** CPU, memory, and network resource consumption tracking

Security Validation

Security Assessment Framework

- **Access Control Verification:** Permission boundary testing and validation
- **Authentication Testing:** Security credential validation and rotation procedures
- **Data Protection:** Sensitive information handling verification and audit procedures
- **Compliance Validation:** Regulatory requirement adherence verification

Development Framework

SDK Integration

Tool development utilizes the pgai-ai-builder SDK which provides:

- **Base Classes:** Standardized tool interfaces and result structures
- **Utility Functions:** Common operations for API integration and data processing

- **Configuration Management:** Environment variable handling and security validation
- **Testing Framework:** Unit testing utilities for tool validation

SDK Documentation: Complete implementation guidance available in the [pgai-ai-builder repository](#) .

Deployment Process

Tool Packaging Tools package as deployable components that include:

- Implementation code using pgai-ai-builder SDK
- Configuration schema defining parameters and requirements
- Documentation including usage examples and security considerations
- Test cases validating functionality and error handling

Operational Considerations

Deployment Management

Tool Lifecycle

- **Version Control:** Systematic tool versioning with backward compatibility management
- **Deployment Automation:** Consistent tool deployment procedures across environments
- **Rollback Procedures:** Automated rollback capabilities for problematic deployments
- **Documentation Maintenance:** Comprehensive documentation updates aligned with tool evolution

Maintenance Procedures

Ongoing Operations

- **Dependency Management:** External system connectivity monitoring and maintenance
- **Performance Optimization:** Continuous performance tuning based on usage patterns
- **Security Updates:** Regular security assessment and vulnerability remediation
- **Capacity Planning:** Resource allocation optimization based on usage growth

Implementation Resources

Development Guides

Specific implementation procedures are covered in dedicated how-to guides:

- *Create a Tool in Gen AI Builder* : Step-by-step tool development using pgai-ai-builder SDK
- *Tool Configuration File* : Configuration schema and deployment procedures
- *Run Tool* : Execution testing and validation procedures

Reference Materials

SDK Documentation

- [pgai-ai-builder SDK](#) : Complete SDK documentation and examples
- Tool API Reference : Technical specification for tool interfaces

Integration Documentation

- *Assistants Reference Manual* : Tool utilization within conversational interfaces
- *Structures Reference Manual* : Tool incorporation in automated processes

Tools transform passive AI systems into active participants in organizational workflows by providing secure, scalable interfaces for external system integration and custom logic execution within the Gen AI Builder ecosystem.

Threads Reference Manual

Threads Reference Manual

Threads provide comprehensive conversation management capabilities within Gen AI Builder, maintaining persistent interaction histories, context preservation, and operational oversight for AI assistant deployments. They serve as the foundation for conversation continuity, quality assurance, and compliance monitoring across production AI applications.

Architectural Function

Threads operate as the conversation persistence layer that captures complete interaction sequences between users and assistants while maintaining associated metadata, system context, and operational details. This persistence enables continuous conversation experiences, systematic quality evaluation, and comprehensive audit capabilities for enterprise AI deployments.

System Integration Framework

Gen AI Builder Integration Threads integrate seamlessly with all conversational components within the Gen AI Builder ecosystem:

- **Assistants:** Maintain conversation continuity across multi-turn interactions
- **Memory Systems:** Provide persistent storage for conversation context and learned preferences
- **Tools:** Record tool execution details and integration outcomes
- **Rulesets:** Track behavioral compliance and policy adherence across interactions

Conversation Processing Pipeline

```

User Input → Assistant Processing → Response Generation → Thread Recording
      ↓           ↓           ↓           ↓
Context Retrieval → Tool Execution → Memory Updates → Audit Trail Creation
  
```

This integration ensures comprehensive capture of all interaction elements while maintaining system performance and user experience quality.

Core Architecture

Thread Structure

Conversation Elements Each thread maintains comprehensive records of interaction components:

Component	Description	Purpose
Message Sequence	Complete user-assistant exchange history	Conversation continuity and context preservation
Metadata Records	Timestamps, identifiers, and system information	Operational tracking and performance analysis
Tool Execution Logs	External system interaction details	Integration monitoring and troubleshooting
Memory State Tracking	Context evolution and preference learning	Personalization and conversation enhancement

Temporal Organization Threads maintain chronological message organization with precise timestamp information, enabling accurate conversation reconstruction and temporal analysis for quality assurance and compliance purposes.

State Management

Context Preservation Threads maintain conversation context across multiple interaction sessions, supporting long-term relationships between users and assistants while ensuring consistent service quality and personalized experiences.

Memory Integration Advanced memory systems utilize thread data for learning user preferences, conversation patterns, and operational context that enhances future interactions while maintaining appropriate privacy boundaries.

Operational Capabilities

Quality Assurance Framework

Conversation Analysis Threads provide systematic access to complete conversation histories enabling comprehensive quality evaluation across multiple dimensions:

- **Response Accuracy:** Verification of factual correctness and source attribution
- **Behavioral Compliance:** Assessment of ruleset adherence and policy compliance
- **User Experience:** Evaluation of conversation flow, satisfaction, and task completion
- **System Performance:** Analysis of response times, tool execution, and operational efficiency

Performance Monitoring Systematic thread analysis supports continuous improvement through pattern identification, common issue detection, and optimization opportunity discovery across assistant deployments.

Compliance and Audit Support

Regulatory Compliance Threads provide comprehensive audit trails supporting regulatory requirements across industries including financial services, healthcare, and data protection regulations.

Documentation Standards Complete interaction records support documentation requirements for quality systems, compliance frameworks, and operational oversight procedures mandated by organizational policies.

Debugging and Troubleshooting

Issue Investigation Thread analysis enables systematic investigation of operational issues including:

- **Behavioral Anomalies:** Identification of unexpected assistant responses or policy violations
- **Integration Failures:** Analysis of tool execution problems and external system connectivity issues
- **Performance Degradation:** Investigation of response latency or quality reduction patterns

- **User Experience Issues:** Examination of conversation breakdowns or satisfaction problems

Root Cause Analysis Detailed thread information supports comprehensive root cause analysis enabling systematic resolution of operational issues and prevention of recurring problems.

Management Operations

Thread Organization

Search and Discovery Comprehensive search capabilities enable efficient thread location through multiple criteria:

- **Temporal Filtering:** Date ranges and activity periods
- **Content Search:** Keyword and topic identification
- **Metadata Filtering:** Assistant types, user categories, and interaction patterns
- **Performance Criteria:** Response quality, completion rates, and satisfaction metrics

Categorization Systems Systematic thread organization supports efficient management through classification schemes aligned with operational requirements and analysis objectives.

Lifecycle Management

Retention Policies Thread management implements configurable retention policies balancing operational requirements with privacy obligations and storage optimization considerations.

Archive Procedures Systematic archival processes ensure long-term data preservation for compliance requirements while optimizing active system performance through appropriate data lifecycle management.

Access Control Framework

Permission Management Comprehensive access control ensures appropriate thread visibility aligned with organizational roles, privacy requirements, and operational responsibilities.

Audit Trail Integrity Thread modification tracking maintains complete audit trails supporting compliance requirements and operational integrity verification procedures.

Implementation Patterns

Enterprise Quality Assurance

Organizations implement systematic thread analysis procedures supporting comprehensive quality management across assistant deployments while maintaining operational efficiency and compliance requirements.

Quality Assessment Workflows

- Regular sampling procedures for conversation quality evaluation
- Systematic compliance checking aligned with regulatory requirements
- Performance trend analysis supporting continuous improvement initiatives
- User feedback integration enhancing service quality and satisfaction

Compliance Monitoring

Regulated industries utilize thread capabilities for comprehensive compliance monitoring supporting audit requirements and regulatory oversight obligations.

Compliance Framework Integration

- Automated policy adherence checking and violation detection
- Comprehensive documentation supporting regulatory reporting requirements
- Risk assessment procedures identifying potential compliance issues
- Remediation tracking ensuring systematic issue resolution

User Experience Optimization

Product teams leverage thread analysis for systematic user experience enhancement through pattern identification and optimization opportunity discovery.

Experience Enhancement Procedures

- Conversation flow analysis identifying improvement opportunities
- User satisfaction correlation with system performance characteristics
- Feature utilization tracking informing development priorities
- Personalization effectiveness measurement supporting system enhancement

Technical Considerations

Performance Characteristics

Storage Efficiency Thread storage optimization balances comprehensive data capture with system performance requirements through intelligent compression and archival strategies.

Query Performance Search and analysis operations maintain responsive performance across large thread collections through optimized indexing and query execution strategies.

Integration Requirements

System Compatibility Thread systems integrate seamlessly with existing organizational infrastructure including identity management, audit systems, and compliance platforms.

Data Export Capabilities Comprehensive export functionality supports integration with external analysis tools, compliance systems, and quality management platforms.

Operational Best Practices

Thread Analysis Procedures

Systematic Review Methods

- Regular sampling schedules ensuring comprehensive quality coverage
- Standardized evaluation criteria supporting consistent assessment
- Documentation procedures maintaining comprehensive analysis records
- Escalation protocols addressing significant issues or policy violations

Continuous Improvement Integration

- Analysis findings integration with system enhancement procedures
- Pattern identification supporting proactive issue prevention
- Optimization opportunity documentation informing development priorities
- Success metric tracking demonstrating improvement effectiveness

Privacy and Security

Data Protection Thread management implements comprehensive privacy protections including data minimization, access control, and retention limitation aligned with organizational policies and regulatory requirements.

Security Framework Comprehensive security measures protect thread data integrity and confidentiality while supporting operational access requirements and compliance obligations.

Implementation Resources

Management Procedures

Thread Administration

- *How-To View and Manage Threads in Gen AI Builder in Hybrid Manager* : Comprehensive thread access and analysis procedures
- *Threads Reference Manual* : Systematic evaluation techniques and quality assessment approaches

Integration Documentation

System Connectivity

- *Assistants Reference Manual* : Thread integration with conversational systems
- *Assistants Reference Manual* : Context preservation and learning capabilities
- *Rulesets Manual* : Behavioral monitoring and policy enforcement

SDK reference

- *Conversation Memory*
- Conversation Memory Drivers

Threads provide essential conversation management infrastructure that enables persistent interactions, comprehensive quality assurance, and systematic compliance monitoring across all Gen AI Builder applications while supporting continuous improvement and operational excellence.

Data Lake Reference Manual

Data Lake Reference Manual

The Data Lake provides foundational object storage infrastructure for Gen AI Builder operations, serving as the persistent storage backend for content ingestion, workflow artifacts, and system components. This storage layer enables the complete AI Factory content pipeline from initial data source ingestion through production assistant deployment.

Architectural Foundation

The Data Lake functions as the central storage repository that supports all Gen AI Builder operations requiring persistent data management. It provides the storage infrastructure necessary for content processing workflows, component deployment, and operational artifact management across the entire AI Factory ecosystem.

System Integration Role

Gen AI Builder Pipeline Integration The Data Lake operates as a critical infrastructure component within the complete content processing pipeline:

Data Sources → Data Lake → Libraries → Knowledge Bases → Retrievers → Assistants

This integration enables systematic content flow from initial ingestion through production deployment while maintaining data persistence and accessibility requirements.

Component Storage Functions

- **Content Staging:** Temporary storage for content during processing and transformation workflows
- **Artifact Management:** Persistent storage for deployment packages, configurations, and system components
- **Workflow Support:** Intermediate storage for multi-stage processing operations
- **Component Repository:** Storage for Structures, Tools, and custom implementation packages

Storage Architecture

Infrastructure Requirements

Object Storage Compatibility The Data Lake requires S3-compatible object storage infrastructure supporting standard object operations, access control mechanisms, and Cross-Origin Resource Sharing (CORS) configuration for console integration.

Supported Storage Backends

- Amazon S3 with appropriate bucket policies and IAM permissions
- Google Cloud Storage with proper access controls and service account configuration
- Azure Blob Storage with container-level permissions and authentication setup
- S3-compatible storage systems supporting standard API operations

Access Control Framework

Permission Requirements Data Lake operations require comprehensive permissions enabling content upload, retrieval, and management operations across different organizational roles and system components.

Security Isolation Dedicated Data Lake configuration ensures proper separation between AI Factory operations and other organizational storage requirements, supporting both security isolation and operational efficiency.

Operational Functions

Content Management

Data Source Integration The Data Lake serves as the storage backend for Data Lake-type data sources, enabling direct content upload and management through Gen AI Builder interfaces.

Processing Workflow Support Content transformation workflows utilize Data Lake storage for intermediate processing stages, enabling complex multi-step operations while maintaining data persistence and recoverability.

Component Deployment

Structure Storage Griptape Structures package as deployment artifacts stored within the Data Lake, enabling systematic component management and version control for complex workflow implementations.

Tool Repository Custom tools and extensions utilize Data Lake storage for deployment packages, configuration files, and runtime dependencies required for operational functionality.

Workflow Orchestration

Temporary Artifact Management Processing workflows generate intermediate artifacts and temporary data requiring persistent storage during multi-stage operations and complex transformation processes.

State Persistence Long-running operations and complex workflows utilize Data Lake storage for state management and checkpoint operations supporting reliable processing and recovery capabilities.

Configuration Requirements

Infrastructure Setup

Storage Provisioning Data Lake configuration requires dedicated object storage allocation with appropriate capacity planning, access control configuration, and performance characteristics suitable for AI workload requirements.

Network Configuration CORS configuration enables browser-based console operations while maintaining appropriate security boundaries for programmatic access and automated processing workflows.

Access Management

Credential Requirements Data Lake access requires appropriate authentication credentials with sufficient permissions for content management, component deployment, and workflow operations across all supported use cases.

Permission Scoping Access control configuration should implement least-privilege principles while enabling necessary operations for content management, component deployment, and system functionality.

Performance Considerations

Storage Optimization

Access Pattern Optimization Data Lake configuration should consider expected access patterns including content upload frequencies, processing workflow requirements, and component deployment patterns to optimize performance characteristics.

Capacity Planning Storage capacity requirements scale with content volume, component complexity, and workflow artifact generation, requiring systematic capacity planning and monitoring procedures.

Operational Efficiency

Content Organization Systematic organization of Data Lake contents supports efficient operations and simplifies management procedures for content, components, and workflow artifacts.

Cleanup Procedures Regular maintenance procedures ensure optimal storage utilization through systematic cleanup of temporary artifacts, obsolete components, and unnecessary content accumulation.

Integration Patterns

Data Source Connectivity

Direct Upload Integration Data Lake-type data sources provide direct content upload capabilities enabling systematic content ingestion without intermediate processing or transformation requirements.

Workflow Integration Complex data source workflows utilize Data Lake storage for content staging, transformation artifacts, and processing state management during multi-stage ingestion operations.

Component Management

Development Workflow Integration Structure and Tool development workflows utilize Data Lake storage for deployment packages, enabling systematic component management and version control procedures.

Operational Deployment Production component deployment utilizes Data Lake storage for runtime artifacts, configuration files, and dependency management supporting reliable system operations.

Operational Best Practices

Storage Management

Isolation Strategies Dedicated Data Lake configuration ensures appropriate separation between AI Factory operations and other organizational storage requirements, supporting both security and operational efficiency.

Version Management Systematic versioning procedures for stored components and content enable reliable deployment management and rollback capabilities when operational issues arise.

Security Framework

Access Control Implementation Comprehensive access control configuration ensures appropriate permissions for different user types and system components while maintaining security boundaries and operational requirements.

Audit Capabilities Storage access logging and monitoring capabilities support operational oversight and compliance requirements through comprehensive visibility into Data Lake operations.

Implementation Dependencies

Prerequisites

Data Lake configuration represents a foundational requirement for Gen AI Builder functionality. Libraries, Knowledge Bases, and assistant operations depend on properly configured Data Lake infrastructure for content management and system operations.

Configuration Resources

Setup Procedures

- *Configure the Data Lake in Gen AI Builder* : Comprehensive setup procedures for object storage integration
- *Configure Data Sources in Gen AI Builder* : Data Lake integration with content ingestion workflows

Integration Documentation

- *Libraries Manual* : Content management workflow integration
- *Structures Reference Manual* : Structure and Tool deployment procedures

The Data Lake provides essential infrastructure that enables all content management, component deployment, and workflow orchestration capabilities within Gen AI Builder, serving as the foundational storage layer for the complete AI Factory ecosystem.

Libraries Manual

Libraries provide the foundational data management infrastructure for knowledge-driven AI applications within Gen AI Builder. They systematically organize, process, and optimize diverse data sources to support retrieval-augmented generation workflows that power intelligent assistants and automated systems.

Architectural Function

Libraries serve as the data orchestration layer between raw information sources and intelligent applications. They transform unstructured content from diverse origins into searchable, semantically-indexed knowledge resources that assistants and other AI components can reliably access for accurate information retrieval.

Data Processing Pipeline

The library system implements a structured data transformation workflow:

```
Data Sources → Content Ingestion → Processing & Indexing → Knowledge Bases →
↔ Retrieval Interface
```

This architecture ensures consistent data quality while maintaining operational efficiency across diverse content types and organizational requirements.

System Integration

Gen AI Builder Components

- **Knowledge Bases:** Receive processed content with semantic indexing and metadata enrichment
- **Retrievers:** Execute optimized search operations across library-managed content
- **Assistants:** Access curated knowledge through standardized retrieval interfaces
- **Data Lakes:** Integrate with large-scale data processing workflows via AI Accelerator Pipelines

AI Factory Infrastructure

- **Vector Engine:** Provides high-performance semantic search capabilities within Postgres infrastructure
- **AI Accelerator Pipelines:** Enables automated content processing and knowledge base maintenance
- **Model Serving:** Supplies embedding models for semantic indexing operations

Core Components

Data Source Integration

Libraries connect to diverse data repositories through standardized interfaces that abstract source-specific complexity while maintaining comprehensive content access capabilities.

Supported Source Types

Source Type	Integration Method	Content Processing
Web Pages	URL crawling and scraping	HTML parsing with content extraction
Document Repositories	API integration or file system access	Format-specific processing (PDF, DOCX, etc.)
Database Systems	Direct database connectivity	Query-based content extraction
Cloud Storage	Object storage APIs (S3, GCS, Azure)	Batch processing with meta-data preservation
Collaborative Platforms	API-based integration (Confluence, SharePoint)	Structured content extraction with hierarchy

Content Processing Framework

Data ingestion implements systematic content transformation procedures:

1. **Format Detection:** Automatic identification of content types and structures
2. **Content Extraction:** Format-specific parsing that preserves semantic meaning
3. **Metadata Enrichment:** Addition of contextual information including source attribution, timestamps, and classification tags
4. **Quality Validation:** Content accuracy verification and completeness checking
5. **Semantic Indexing:** Vector embedding generation for similarity-based search operations

Knowledge Base Architecture

Knowledge bases provide the operational interface between processed library content and retrieval systems. They implement optimized storage and indexing strategies that balance search performance with resource efficiency.

Storage Optimization

- Vector embeddings stored within Postgres infrastructure using Vector Engine capabilities
- Content chunking strategies optimized for retrieval accuracy and model context windows
- Hierarchical indexing that preserves document structure and relationships
- Metadata indexing for filtered search operations and access control enforcement

Search Capabilities

- Semantic similarity search using vector embeddings
- Hybrid search combining vector similarity with keyword matching
- Filtered search supporting metadata-based content selection
- Reranking algorithms that optimize result relevance for specific query types

Retrieval Interface

Retrievers define the operational parameters for knowledge access, providing configurable search strategies that optimize for different application requirements and content characteristics.

Configuration Parameters

Parameter	Purpose	Typical Range
Similarity Threshold	Controls result relevance filtering	0.6 - 0.9
Result Limit (Top-K)	Defines maximum results returned	3 - 20
Context Window	Manages content length for language models	1000 - 8000 tokens
Reranking Strategy	Optimizes result ordering for specific use cases	None, semantic, hybrid

Advanced Retrieval Strategies

- Multi-query expansion for comprehensive information coverage
- Contextual filtering based on user permissions and data classification
- Dynamic threshold adjustment based on query characteristics
- Citation and provenance tracking for source attribution

Implementation Patterns

Enterprise Knowledge Management

Organizations implement comprehensive knowledge management systems that unify information across departments and business functions through centralized library infrastructure.

Architecture Characteristics

- Multi-source integration across diverse organizational systems
- Role-based access control aligned with organizational hierarchies
- Automated content updates synchronized with source system changes
- Compliance frameworks ensuring regulatory requirement adherence

Typical Implementation

```
# Library Configuration Example

library_config:
  name: "enterprise_knowledge"
  data_sources:
    - type: "confluence"
      connection: "corporate_confluence"
      spaces: ["hr_policies", "it_procedures", "finance_guidelines"]
    - type: "sharepoint"
      connection: "corporate_sharepoint"
      sites: ["project_documentation", "training_materials"]
  processing:
    chunk_size: 1000
    overlap: 100
    embedding_model: "organizational_embeddings"
  knowledge_base:
    name: "enterprise_kb"
    search_strategy: "hybrid"
    access_control: "rbac_enabled"
```

Customer Support Systems

Customer-facing applications leverage libraries to provide consistent, accurate information delivery through conversational interfaces while maintaining comprehensive coverage of product and policy information.

Design Considerations

- Content accuracy verification procedures for customer-facing information
- Update synchronization ensuring consistency between internal and external documentation
- Performance optimization for high-concurrency customer interactions
- Integration with existing support systems and escalation workflows

Research and Development Support

Technical teams utilize libraries to aggregate research materials, technical documentation, and project information for enhanced productivity and knowledge sharing across development initiatives.

Specialized Features

- Technical document processing optimized for code examples and diagrams
- Version control integration for tracking document evolution
- Cross-reference linking that maintains relationships between related materials
- Export capabilities for integration with development tools and workflows

Operational Management

Content Lifecycle

Ingestion Procedures Libraries implement systematic content ingestion that ensures data quality while maintaining operational efficiency:

1. **Source Monitoring:** Automated detection of content changes and additions
2. **Processing Queues:** Managed processing workflows that handle varying content volumes
3. **Quality Assurance:** Validation procedures ensuring content accuracy and completeness
4. **Index Updates:** Incremental updates that maintain search performance during content changes

Maintenance Operations

- Regular content validation ensuring continued accuracy and relevance
- Performance optimization including index restructuring and storage optimization
- Security updates maintaining access control accuracy and compliance requirements
- Backup procedures preserving both content and configuration information

Performance Optimization

Search Performance Libraries implement multiple optimization strategies that ensure responsive search operations across large content collections:

- **Index Optimization:** Regular index maintenance that optimizes search performance
- **Caching Strategies:** Intelligent caching of frequently accessed content and search results
- **Resource Allocation:** Dynamic resource scaling based on usage patterns and performance requirements

- **Query Optimization:** Automatic query enhancement that improves search accuracy and efficiency

Scalability Management

- Horizontal scaling capabilities that support growing content volumes and user bases
- Load balancing across processing resources ensuring consistent performance
- Storage optimization strategies that balance performance with cost efficiency
- Monitoring frameworks that provide visibility into system performance and capacity utilization

Configuration Framework

Data Source Configuration

Data source connections require careful configuration that balances comprehensive content access with security requirements and operational efficiency.

Authentication and Authorization

```
# Example Data Source Configuration

data_source:
  type: "confluence"
  connection:
    base_url: "https://company.atlassian.net"
    authentication:
      method: "api_token"
      credentials: "service_account"
    access_scope:
      spaces: ["public_docs", "team_docs"]
      permissions: "read_only"
```

Content Selection

- Filter configuration that defines content inclusion criteria
- Update frequency settings that balance freshness with resource consumption
- Processing parameters optimized for specific content types and organizational requirements

Knowledge Base Configuration

Knowledge base setup requires systematic configuration that optimizes search performance while maintaining content quality and accessibility.

Processing Parameters

- Chunking strategies that balance context preservation with search granularity
- Embedding model selection aligned with organizational content characteristics
- Metadata extraction rules that capture relevant contextual information
- Quality validation thresholds that ensure content accuracy and completeness

Retrieval Configuration

Retriever setup defines the operational characteristics of knowledge access, requiring careful tuning that balances search accuracy with performance requirements.

Search Strategy Configuration

```
# Retriever Configuration Example

retriever_config:
  name: "enterprise_retriever"
  search_strategy: "hybrid"
  parameters:
    similarity_threshold: 0.75
    max_results: 10
    reranking: "enabled"
    context_optimization: "automatic"
  access_control:
    user_filtering: "enabled"
    content_classification: "enforced"
```

Quality Assurance

Content Validation

Libraries implement comprehensive validation procedures that ensure content accuracy and completeness across the entire data processing pipeline.

Validation Framework

- Source content accuracy verification through automated and manual review processes
- Processing quality assessment ensuring semantic meaning preservation during transformation
- Search result validation confirming retrieval accuracy and relevance
- User feedback integration that enables continuous quality improvement

Performance Monitoring

Operational Metrics

- Search response times and accuracy measurements
- Content processing throughput and error rates
- System resource utilization and capacity planning metrics
- User satisfaction and system effectiveness measurements

Continuous Improvement

- Performance trend analysis that identifies optimization opportunities
- Content quality assessment enabling targeted improvements
- User behavior analysis informing search strategy refinements
- System optimization procedures that maintain peak performance

Security and Compliance

Access Control

Libraries implement comprehensive access control frameworks that ensure appropriate information exposure while maintaining operational efficiency and user experience quality.

Security Framework

- User authentication integration with organizational identity systems
- Content-level access control respecting organizational data classification
- Audit logging providing comprehensive visibility into information access patterns
- Privacy controls ensuring appropriate handling of sensitive information

Compliance Integration

Regulatory Compliance

- Data retention policies aligned with organizational and regulatory requirements
- Privacy protection mechanisms ensuring appropriate personal information handling
- Audit trail maintenance supporting compliance verification and reporting
- Data lineage tracking providing comprehensive visibility into information processing

Getting Started

Prerequisites

Library implementation requires foundational infrastructure including Gen AI Builder installation, appropriate compute resources for content processing, and network connectivity to organizational data sources.

Initial Configuration

1. **Data Source Assessment:** Evaluate available content sources and access requirements
2. **Processing Configuration:** Define content processing parameters based on organizational needs
3. **Knowledge Base Setup:** Configure search strategies and performance optimization
4. **Retrieval Testing:** Validate search accuracy and performance characteristics

Implementation Resources

Configuration Guides

- *Configure Data Sources in Gen AI Builder* : Comprehensive source integration procedures
- *Knowledge Bases Reference Manual* : Search optimization and performance tuning
- *Retrievers Reference Manual* : Search strategy definition and parameter optimization

Integration Documentation

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)* : Ecosystem connectivity patterns
- *Hybrid Manager Deployment* : Infrastructure integration procedures

SDK reference

- Embedding Drivers
 - Vector Store Drivers
 - Data Loaders
-

Libraries provide the foundational data management infrastructure that enables knowledge-driven AI applications through systematic content organization, processing optimization, and intelligent retrieval capabilities within the Gen AI Builder ecosystem.

Explainers

Best practices for Hybrid Knowledge Bases

Who is this for

Platform users creating Hybrid Knowledge Bases to power advanced AI applications combining structured and unstructured data. Typical users include developers, AI architects, data engineers, and content managers working with product catalogs, customer profiles, or mixed datasets.

What you will accomplish

You will learn best practices for designing Hybrid Knowledge Bases in Gen AI Builder:

- How to select structured and unstructured columns
- How to design a schema that supports powerful AI queries
- How to avoid common pitfalls in Hybrid Knowledge Base design

Why design Hybrid Knowledge Bases carefully

- Hybrid Knowledge Bases allow you to combine **structured filters** (exact matches, ranges, facets) with **semantic search** (vector-based search).
- A well-designed Hybrid Knowledge Base enables more accurate, flexible, and explainable AI responses.
- Poor schema design can lead to inefficient queries, incomplete search results, or wasted processing.

For background, see:

- *Knowledge Bases Reference Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Retrieval Augmented Generation (RAG)*
- *Embeddings explained*

To see how Hybrid Knowledge Bases are used in **Hybrid Manager deployments**, visit:

- [Hybrid Manager: Using Gen AI Builder](#)

Complexity and time to complete

- **Complexity:** Moderate — schema design requires understanding your data and use cases.
- **Estimated time:** 15–30 minutes per Hybrid Knowledge Base.

Designing Hybrid Knowledge Bases

Understand your use case

Start by understanding what types of queries you want your AI application to support.

Examples:

- “Show products under \$500 in category ‘Outdoor’ and rank by relevance to search terms.”
- “Find policies for region ‘EU’ that mention data retention.”

Selecting structured columns

Structured columns support:

- Exact match filters (e.g., `region = 'EU'`)
- Range queries (e.g., `price BETWEEN 100 AND 500`)
- Faceted search (e.g., list all `categories`)

Best practices:

- Use **short, unambiguous column names** that match your Library schema exactly.
- Choose data types carefully:
- `Text` for identifiers and categories (SKU, region, category)
- `Number` for numeric ranges (price, rating)
- `Date` for date-based filtering (release date, expiration date)
- `Boolean` for simple yes/no filters (isActive, inStock)
- Do not include freeform text fields as structured columns — they should go in unstructured columns.

Selecting unstructured columns

Unstructured columns support:

- Full-text semantic search
- Retrieval-Augmented Generation (RAG) queries

Best practices:

- Select columns containing rich text that will benefit from embedding:
- Descriptions
- Reviews
- Policies
- FAQ answers
- Detailed notes
- Avoid columns with redundant or trivial text — this can degrade search quality.
- Be intentional about **how many** unstructured columns you define:
- Too many dilute search results.
- Too few limit context richness.

Example schema design

Product catalog Hybrid KB

Structured columns:

- SKU (Text)
- Category (Text)
- Price (Number)
- AvailableSince (Date)
- InStock (Boolean)

Unstructured columns:

- ProductName
- Features
- DetailedDescription

Common pitfalls

Mismatched column names

- The column names you define **must match exactly** the column names in your source Library (CSV headers, Google Sheet columns).
- Misspelled names or casing differences will cause ingestion errors.

Poor type selection

- Do not mark numeric fields as Text or vice versa.
- Use Date types correctly to support proper date queries.

Overloading structured columns

- Do not add long freeform text columns as structured — this will degrade filtering performance.

Underusing unstructured columns

- If your unstructured text is spread across multiple columns (e.g., FAQ question, FAQ answer), consider combining them with a Structure before ingestion, or clearly select both as unstructured.

Example scenario

You are designing a Hybrid Knowledge Base for your **Product Catalog**.

Example configuration:

- **Structured Columns:**
 - SKU (Text)
 - Category (Text)
 - Price (Number)
 - AvailableSince (Date)

- **Unstructured Columns:**

- `ProductName`
- `DetailedDescription`

Resulting queries supported:

- Filter by `Category` and `Price` range.
- Sort by `AvailableSince`.
- Perform semantic search across product names and descriptions.

Troubleshooting

Incomplete search results

- Verify that unstructured columns were selected and indexed correctly.
- Check that structured filters in queries are not too restrictive.

Filter not working

- Verify that the column was marked as structured and with the correct data type.
- Check that the Library schema and column names match exactly.

Ingestion errors

- Check that all column names match your source Library exactly.
- Verify that the data types of columns align with your intended schema.

Related topics

- *Knowledge Bases Reference Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Retrieval Augmented Generation (RAG)*
- *Embeddings explained*
- *Structures Reference Manual*
- *Create a Knowledge Base in Gen AI Builder*
- *Manage Knowledge Bases in Gen AI Builder*
- *Hybrid Manager: Using Gen AI Builder*

How to prepare data and kb

Gen AI How-To Guides

Use these guides to build and manage AI-powered applications in AI Factory's Gen AI capabilities.

These how-to guides cover day-to-day tasks for configuring your AI Agents (Assistants and Structures), integrating external data sources, managing Rulesets, and more.

Where to start

- *Configure the Data Lake in Gen AI Builder*
- *Create a Knowledge Base in Gen AI Builder*
- *Create a Retriever in Gen AI Builder*
- *Create a Ruleset in Gen AI Builder*
- *Create a Structure in Gen AI Builder*
- *Create a Tool in Gen AI Builder*

Working with Data Sources

- *Configure an Atlassian Confluence data source*
- *Configure a Custom data source*
- *Configure a Data Lake data source*
- *Configure a Google Drive data source*
- *Configure an Amazon S3 data source*
- *Configure a Web Page data source*

Best practices and management

- *Best practices for Hybrid Knowledge Bases*
- *Manage Knowledge Bases in Gen AI Builder*
- *How-To View and Manage Threads in Gen AI Builder in Hybrid Manager*

Related concepts

- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Structures Reference Manual*
- *Tools Reference Manual*
- *Knowledge Bases Reference Manual*
- *Retrievers Reference Manual*
- *Rulesets Manual*
- *Threads Reference Manual*

SDK reference

- SDK Overview
- *Structures Reference Manual*
- *Tools Reference Manual*
- *Vector databases and semantic search*

Create a Knowledge Base in Gen AI Builder

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users configuring **Knowledge Bases** in **Gen AI Builder** in **Hybrid Manager AI Factory**. Typical users include developers, AI architects, and business owners managing Knowledge Bases for production AI applications.

What you will accomplish

You will create a **Knowledge Base** by selecting one or more Libraries as its content source.

You can configure:

- **Fully-managed vector stores**
- **Self-managed Postgres vector stores**
- **Hybrid Knowledge Bases** combining structured and unstructured data

You will prepare your Knowledge Base for use in:

- AI Assistants
- RAG pipelines
- Semantic search applications

Why use Knowledge Bases

Knowledge Bases provide the **queryable foundation** for Gen AI applications:

- Power **semantic search** and **Retrieval-Augmented Generation (RAG)**.
- Enable LLM-based agents to retrieve content grounded in your data.
- Combine structured metadata with rich unstructured text for advanced use cases.
- Support refreshable pipelines — when source Libraries are updated, Knowledge Bases can be re-synced.

For background, see:

- *Knowledge Bases Reference Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Retrieval Augmented Generation (RAG)*
- *Embeddings explained*
- *Structures Reference Manual*

To see how Knowledge Bases are used in **Hybrid Manager deployments**, visit:

- Hybrid Manager: Using Gen AI Builder

Complexity and time to complete

- **Complexity:** Low to moderate (depends on Knowledge Base type selected)
- **Estimated time:** 5–15 minutes

When to create a Knowledge Base

- When preparing content for use in **Assistants**.
- To enable **RAG pipelines**.
- To deliver semantic search experiences.
- To combine structured filters with AI-driven search (Hybrid KB).
- After creating and validating **Libraries** with your source content.

How to create a Knowledge Base

1. Navigate to Knowledge Bases

- In Gen AI Builder console, go to **Knowledge Bases**.

2. Create a new Knowledge Base

- Click **Create Knowledge Base**.

3. Configure common fields

Name (required)

- Provide a clear, unique name.
- Example: Company Policy KB

Description (optional)

- Describe the Knowledge Base's purpose or intended use.

4. Select Knowledge Base type

Gen AI Builder supports **three Knowledge Base types**:

PG.AI - Fully-Managed Vector Store

- Create a vector-based Knowledge Base with a fully managed backend.
- Optimized for unstructured text and general-purpose semantic search.

Configure:

- **Libraries:** Select one or more Libraries.
- **Example:** HR Policy Documents Library, Employee Handbook Library

Example

- **Name:** Company Policy KB
- **Libraries selected:** HR Policy Documents Library, Employee Handbook Library

PG.AI Database - Self-Managed

- Connect to your own Postgres-compatible database with AI/vector extensions.
- Useful when you want full control over your vector store (ex: HM-managed Postgres + AIDB).

Configure:

- **Connection String:** Full connection string to your database.
- **Example:** postgresql://user:password@host:port/dbname
- **Password:** Database password.
- **Libraries:** Select one or more Libraries.

Example

- **Name:** Tech Docs Self-Managed KB
- **Connection String:** postgresql://user:password@host:5432/techdocs
- **Password:** *****
- **Library selected:** API Documentation Library

PG.AI Hybrid Database

- Build a Hybrid Knowledge Base with both **structured filters** and **semantic search**.
- Ideal for catalogs, customer profiles, and multi-attribute search experiences.

Configure:

- **Libraries:** Select one or more Libraries (typically structured data sources such as CSV, Google Sheet).
- **Structured Columns:**
 - Add columns used for filtering.
 - Define name, optional description, data type (Text, Number, Date, Boolean).
- **Unstructured Columns:**
 - Add columns used for semantic search embeddings.
 - Define name and optional description.

Example

- Name: Product Catalog Hybrid KB
- Library selected: Product CSV Library
- Structured Columns:
- SKU (Text)
- Price (Number)
- Unstructured Columns:
- ProductName
- Features

5. Finalize creation

- Click **Create**.
- The system will build and populate your Knowledge Base from the selected Libraries.
- Progress is visible in the **Knowledge Bases** list view.

Example scenario

You want to build a Hybrid Knowledge Base to power product search and filtering in your AI Assistant.

Example configuration:

- Name: Product Catalog Hybrid KB
- Library selected: Product CSV Library
- Structured Columns:
- SKU (Text)
- Price (Number)
- Unstructured Columns:
- ProductName
- Features

Troubleshooting

Missing Libraries

- Verify that your source Libraries are created and populated.
- Hybrid Knowledge Bases require Libraries with defined schemas.

Connection errors (Self-Managed)

- Validate your database connection string and credentials.
- Ensure required extensions (pgvector, AIDB) are installed.
- Confirm that your database is reachable from Gen AI Builder.

Incomplete indexing (Hybrid KB)

- Verify correct columns were selected for structured and unstructured indexing.
- Column names must match the Library schema.

SDK reference

- Embedding Drivers — generate vector embeddings
- Vector Store Drivers — store/query embeddings
- RAG Engines — assemble retrieval and response stages

Related topics

- *Knowledge Bases Reference Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Retrieval Augmented Generation (RAG)*
- *Embeddings explained*
- *Structures Reference Manual*
- *Manage Knowledge Bases in Gen AI Builder*
- *Hybrid Manager: Using Gen AI Builder*

Manage Knowledge Bases in Gen AI Builder

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users managing Knowledge Bases for AI applications in Gen AI Builder. Typical users include developers, AI architects, data engineers, and content managers.

What you will accomplish

You will learn how to manage existing Knowledge Bases in Gen AI Builder. You will perform common management tasks:

- Viewing existing Knowledge Bases
- Editing Knowledge Base settings
- Refreshing or re-syncing Knowledge Bases
- Deleting Knowledge Bases
- Viewing Knowledge Base details

Why manage Knowledge Bases

- Keeping your Knowledge Bases up to date ensures AI applications deliver current and accurate responses.
- Managing your Knowledge Bases helps control storage and cost by cleaning up unused or obsolete Knowledge Bases.
- Monitoring Knowledge Base status and content improves reliability and performance in production AI workflows.

For background, see:

- *Knowledge Bases Reference Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Retrieval Augmented Generation (RAG)*

To see how Knowledge Bases are used in **Hybrid Manager deployments**, visit:

- Hybrid Manager: Using Gen AI Builder

Complexity and time to complete

- **Complexity:** Low.
- **Estimated time:** 2–5 minutes per Knowledge Base.

Viewing existing Knowledge Bases

1. Navigate to **Knowledge Bases** in the Gen AI Builder console.
2. The Knowledge Bases list view displays:
 - **Name:** User-defined name (example: `docs-temp-hcp`).
 - **Created:** Creation date and time.
 - **Updated:** Last updated date and time.
 - **Status:** Operational status (example: `Succeeded`).
 - **Type:** Backend type (example: `GenAI Builder Vector or vector`).
3. Use pagination controls to navigate large lists (example: `Rows per page: 5 1-5 of 8 < >`).

Viewing Knowledge Base details

1. Click on a Knowledge Base name or **View Details** icon to open its detail page.
2. The detail view typically shows:
 - **Name and Description**
 - **Created** and **Updated** timestamps
 - **Status** and backend **Type**
 - **Linked Libraries** or **Data Sources**
 - **Indexing** or **sync status**
 - Recent activity or job history

Editing a Knowledge Base

Note: Edit capabilities may be limited after Knowledge Base creation, depending on type.

1. In the Knowledge Bases list view, select the **Edit** action (typically from an actions menu or edit icon).
2. You may be able to edit:
 - **Description**
 - In some cases, the set of Libraries or Data Sources (depends on type)
 - Configuration for Hybrid KB structured/unstructured columns (if supported)
3. Save changes.

Refreshing or re-syncing a Knowledge Base

1. In the Knowledge Bases list view, select the **Refresh** or **Re-sync** action.
2. The system will:
 - Pull the latest data from linked Libraries.
 - Re-index the Knowledge Base if necessary.

Use this action when source Libraries have been updated and you want to propagate changes to the Knowledge Base.

Deleting a Knowledge Base

Warning: Deleting a Knowledge Base is irreversible. This will remove the Knowledge Base and its content from AI applications.

1. In the Knowledge Bases list view, select the **Delete** action (typically from an actions menu or delete icon).
2. Confirm the deletion when prompted.

This will not delete the source Libraries — only the Knowledge Base.

Troubleshooting

Knowledge Base status shows “Error” or “Failed”

- Click **View Details** and check the activity log or job history.
- If the Knowledge Base is linked to an external database (Self-Managed PG.AI Database), verify that the database connection is healthy.

Refresh operation does not update expected content

- Verify that the linked Libraries have been refreshed and contain the latest data.
- Re-run the Refresh / Re-sync action.
- Check that Libraries are correctly configured with expected Data Sources and Transformations.

Cannot edit certain fields

- Some Knowledge Base types (especially Fully-Managed Vector Stores and Self-Managed DBs) restrict editing of certain configuration fields after creation.
- To change these fields, you may need to create a new Knowledge Base.

Example scenario

You want to refresh and check the status of a Knowledge Base used in production:

1. Navigate to **Knowledge Bases**.
2. Locate `Company Policy KB`.
3. Click **View Details** → confirm recent Libraries were updated.
4. Click **Refresh** → re-sync the Knowledge Base.
5. Monitor status until **Succeeded**.

SDK reference

- Embedding Drivers
- Vector Store Drivers
- RAG Engines

Related topics

- *Knowledge Bases Reference Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Retrieval Augmented Generation (RAG)*
- *Embeddings explained*
- *Structures Reference Manual*
- *Create a Knowledge Base in Gen AI Builder*
- Hybrid Manager: Using Gen AI Builder

Configure an Atlassian Confluence data source

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users who want to ingest content from Atlassian Confluence into Gen AI Builder for use in Knowledge Bases and Retrieval-Augmented Generation (RAG) workflows. Typical users include developers, technical writers, and knowledge managers who maintain institutional knowledge in Confluence.

What you will accomplish

You will configure an Atlassian Confluence data source in Gen AI Builder to ingest pages from one or more Confluence spaces or URLs. The ingested content will then be available for transformation and indexing into Libraries and Knowledge Bases.

Why use a Confluence data source

- Confluence is a common platform for internal technical documentation, policies, engineering notes, and project knowledge.
- Bringing Confluence content into Gen AI Builder makes this valuable institutional knowledge searchable and usable in AI workflows.
- Targeted ingestion of selected spaces or pages provides fine-grained control.

For background on how this content powers downstream AI use cases, see:

- Knowledge Bases overview
- *Retrieval Augmented Generation (RAG)*
- Hybrid Manager: Using Gen AI Builder

Complexity and time to complete

- **Complexity:** Moderate. You need to obtain an Atlassian API Token and understand your Confluence space structure.
- **Estimated time:** 10–15 minutes if API token is already generated.

Key considerations

Data relevance and quality

- Ingest trusted spaces or specific pages with high-quality documentation.
- Ensure content is well structured and uses semantic page layouts to improve AI processing.

Scope of ingestion

- Use space-specific or page-specific URLs when possible to avoid ingesting broad and irrelevant content.
- If you want to ingest multiple spaces, configure multiple data source entries.

Credentials security

- Use an API Token with read-only access when possible.
- Do not use administrator-level tokens unless necessary.

How to configure an Atlassian Confluence data source

1. In the interface where you manage data sources, select **+ Add New Data Source**.
2. Choose **Confluence** from the available data source types.
3. Configure the following fields:
 - **Name:** Provide a clear and unique name. Example: `Engineering Wiki Confluence`.

- **Description** (optional): Add descriptive context for future reference.
4. Connect to Confluence:
 - **Confluence Site URL**: Enter your Confluence site URL.
 - **Example**: `https://mycompany.atlassian.net` or `https://mycompany.atlassian.net/wiki/spaces/DEVKB`
 - **API Token**: Enter a valid Atlassian API Token.
 - **Atlassian Email**: Enter the email address associated with the API token.
 5. (Optional) Configure advanced options:
 - **Scheduled refresh**: Enable this option to automatically refresh the data on a schedule.
 - Provide a cron expression. Example: `0 2 * * *` (daily at 2 AM).
 - **Transform your data**: Enable this option to apply a PG.AI Structure to transform the data during ingestion.
 - Select an existing Structure from the list.
 6. Select **Create** to add the Confluence data source.

Supported content

- Textual data from Confluence pages.
- Content from supported macros (if rendered as text).
- Content from page hierarchies.

For more about how content is indexed and retrieved, see [Embeddings explained](#).

Managing and refreshing the data source

Once created, the Confluence data source can be viewed and managed through the Data Sources interface.

Actions available:

- **Edit**: Modify the data source configuration.
- **Refresh**: Manually trigger a data ingestion job.
- **Delete**: Remove the data source.

You can also review the data job history to monitor ingestion performance and troubleshoot issues.

SDK reference

- Data Loaders
- Web Scraper Drivers

Troubleshooting

Authentication failure

- Verify that the Confluence Site URL, API Token, and Email are correct.
- Confirm that the API Token has sufficient read permissions.
- Check the data job history for specific errors.

Space or page not found

- Double-check the Confluence URL and verify that your account has access to the target spaces or pages.
- If using a page-specific URL, confirm that the page is not restricted.

Example scenario

You want to index content from the “Developer Knowledge Base” space in your company’s Confluence instance.

Example configuration:

- **Name:** Developer KB Confluence Space
- **Confluence Site URL:** `https://mycompany.atlassian.net/wiki/spaces/DEVKB`
- **API Token:** `xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
- **Atlassian Email:** `developer@example.com`
- **Scheduled refresh:** `0 2 * * *` (daily at 2 AM)
- **Transform your data:** Apply a Structure to strip headers and footers and extract core content.

Related topics

- Knowledge Bases overview
- *Embeddings explained*
- *Create a Knowledge Base in Gen AI Builder* <!-- Placeholder link -->
- Hybrid Manager: Using Gen AI Builder

Configure a Google Drive data source

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users who want to ingest files stored in Google Drive into Gen AI Builder for use in Knowledge Bases and Retrieval-Augmented Generation (RAG) workflows. Typical users include developers, data engineers, and business stakeholders who collaborate on project documentation within Google Workspace.

What you will accomplish

You will configure a Google Drive data source in Gen AI Builder to ingest files from selected Google Drive folders. The content will then be available for transformation and indexing into Libraries and Knowledge Bases.

Why use a Google Drive data source

- Many project documents live in collaborative Google Drive folders.
- Google Docs, Slides, and Sheets are frequently used for dynamic team content that is not available elsewhere.
- Directly connecting to Google Drive allows you to capture this valuable knowledge without manual export steps.

For background on how this content powers downstream AI use cases, see:

- Knowledge Bases overview

- *Retrieval Augmented Generation (RAG)*
- Hybrid Manager: Using Gen AI Builder

Complexity and time to complete

- **Complexity:** Moderate. You must complete an OAuth flow and understand your Google Drive folder structure.
- **Estimated time:** 5–10 minutes.

Key considerations

Data relevance and quality

- Ingest curated folders that contain useful, trusted documents.
- Well-written content in Google Docs and well-formatted Google Slides or Sheets contribute to more effective AI responses.

Scope of ingestion

- Select folders carefully during the OAuth authorization flow.
- Avoid ingesting broad folders (such as entire My Drive or Shared Drive roots) unless intentional.

Permissions scope

- Review the permissions requested during the OAuth process.
- If using Google Workspace, confirm that your organization allows third-party app access as required.

How to configure a Google Drive data source

1. In the interface where you manage data sources, select **+ Add New Data Source**.
2. Choose **Google Drive** from the available data source types.
3. Configure the following fields:
 - **Name:** Provide a clear and unique name. Example: `Project Alpha GDrive Docs`.
 - **Description** (optional): Add descriptive context for future reference.
4. Connect to Google Drive:
 - Select **Connect With Google** to initiate the OAuth 2.0 authorization flow.
 - Sign in to your Google Account and grant the requested permissions.
 - Follow on-screen prompts to select the files or folders to ingest.
5. (Optional) Configure advanced options:
 - **Scheduled refresh:** Enable this option to automatically refresh the data on a schedule.
 - Provide a cron expression. Example: `0 2 * * *` (daily at 2 AM).
 - **Transform your data:** Enable this option to apply a PG.AI Structure to transform the data during ingestion.
 - Select an existing Structure from the list.
6. Select **Create** to add the Google Drive data source.

Supported file types

- Google Workspace files: Docs, Sheets, Slides (converted to text).
- PDF
- CSV
- Markdown
- Most text-based file types.

For more about how content is indexed and retrieved from these files, see [Embeddings explained](#).

Managing and refreshing the data source

Once created, the Google Drive data source can be viewed and managed through the Data Sources interface.

Actions available:

- **Edit:** Modify the data source configuration.
- **Refresh:** Manually trigger a data ingestion job.
- **Delete:** Remove the data source.

You can also review the data job history to monitor ingestion performance and troubleshoot issues.

SDK reference

- Data Loaders
- File Manager Drivers

Troubleshooting

Authentication failed

- Check your Google Account and try the OAuth flow again.
- Verify that your organization allows third-party app access.
- Check browser settings and pop-up blockers if the OAuth flow does not complete.

Files or folders not visible

- Verify that you have access to the target files or folders in Google Drive.
- If using a Shared Drive, ensure that the selected files are visible and accessible to your account.

Example scenario

You want to index project planning documents stored in a shared Google Drive folder.

Example configuration:

- **Name:** Project Plans & Presentations GDrive
- **OAuth flow:** Completed successfully.
- **Selected folder:** Active Projects Q2
- **Scheduled refresh:** 0 2 * * * (daily at 2 AM)

- **Transform your data:** Apply a Structure to summarize Google Docs content and extract key sections.

Related topics

- Knowledge Bases overview
- *Embeddings explained*
- *Create a Knowledge Base in Gen AI Builder*
- Hybrid Manager: Using Gen AI Builder

Configure an Amazon S3 data source

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users who want to ingest files stored in Amazon S3 buckets into Gen AI Builder for use in Knowledge Bases and Retrieval-Augmented Generation (RAG) workflows. Typical users include developers, data engineers, and cloud architects managing enterprise data pipelines.

What you will accomplish

You will configure an Amazon S3 data source in Gen AI Builder to ingest files from one or more S3 URIs. The ingested content will be available for transformation and indexing into Libraries and Knowledge Bases.

Why use an Amazon S3 data source

- S3 is a common storage platform for enterprise content such as PDFs, CSVs, and documents.
- Directly connecting to S3 allows you to leverage existing cloud storage without duplicating data.
- Precise targeting of folders or files ensures high efficiency and control over the ingested content.

For background on how this content powers downstream AI use cases, see:

- Knowledge Bases overview
- *Retrieval Augmented Generation (RAG)*
- Hybrid Manager: Using Gen AI Builder

Complexity and time to complete

- **Complexity:** Moderate. You need to securely configure AWS credentials and understand your S3 URI structure.
- **Estimated time:** 10–15 minutes if IAM permissions are already configured.

Key considerations

Data relevance and quality

- Ingest trusted, curated files that contribute meaningfully to your AI application.
- Well-structured, clearly written documents lead to better downstream AI results.

Scope of ingestion

- Use targeted S3 URIs to ingest only the content you need.
- Avoid specifying root-level prefixes that could ingest unnecessary or sensitive content.

Credentials security

- Use IAM credentials with **least privilege**—grant read-only access only to the specific S3 resources Gen AI Builder needs.
- Do not use over-permissioned root or admin keys.

Cost awareness

- Be mindful of S3 data transfer and retrieval costs, especially for large ingestion jobs or frequent refresh schedules.

How to configure an Amazon S3 data source

1. In the interface where you manage data sources, select + **Add New Data Source**.
2. Choose **Amazon S3** from the available data source types.
3. Configure the following fields:
 - **Name**: Provide a clear and unique name. Example: `Marketing Assets S3`.
 - **Description** (optional): Add descriptive context for future reference.
4. Connect to S3:
 - **AWS Access Key**: Enter your AWS Access Key ID.
 - **AWS Secret Key**: Enter your AWS Secret Access Key.
 - **URIs**: Provide one or more S3 URIs for objects or prefixes.
 - Example: `s3://bucket-name/key_name`
 - Select **Add URI** to add more URIs.
5. (Optional) Configure advanced options:
 - **Scheduled refresh**: Enable this option to automatically refresh the data on a schedule.
 - Provide a cron expression. Example: `0 2 * * *` (daily at 2 AM).
 - **Transform your data**: Enable this option to apply a PG.AI Structure to transform the data during ingestion.
 - Select an existing Structure from the list.
6. Select **Create** to add the Amazon S3 data source.

Supported file types

- PDF
- CSV
- Markdown
- Most text-based file types

For more about how content is indexed and retrieved from these files, see [Embeddings explained](#).

Managing and refreshing the data source

Once created, the Amazon S3 data source can be viewed and managed through the Data Sources interface.

Actions available:

- **Edit:** Modify the data source configuration.
- **Refresh:** Manually trigger a data ingestion job.
- **Delete:** Remove the data source.

You can also review the data job history to monitor ingestion performance and troubleshoot issues.

SDK reference

- File Manager Drivers
- Data Loaders

Troubleshooting

Access denied

- Verify that your IAM user or role has the necessary `s3:GetObject` and (if applicable) `s3:ListBucket` permissions.
- Review your bucket policies and S3 URIs for correctness.
- Check the data job history for specific permission-related errors.

Bucket or key not found

- Double-check S3 URIs for correctness.
- Confirm that the specified keys or prefixes exist and are accessible to your IAM user or role.

Example scenario

You want to index quarterly financial reports stored in an S3 folder.

Example configuration:

- **Name:** Financial Reports S3 Archive
- **AWS Access Key:** AKIAxxxxxxxxxxxxxxxxxxxx
- **AWS Secret Key:** xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
- **URI:** `s3://my-company-financials/quarterly-reports/pdf/`
- **Scheduled refresh:** `0 2 * * *` (daily at 2 AM)
- **Transform your data:** Apply a Structure to summarize PDFs and extract key financial figures.

Related topics

- Knowledge Bases overview
- *Embeddings explained*
- *Create a Knowledge Base in Gen AI Builder* <!-- Placeholder link -->
- Hybrid Manager: Using Gen AI Builder

Configure a Web Page data source

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users who want to bring web content into Gen AI Builder for use in Knowledge Bases and Retrieval-Augmented Generation (RAG) workflows. Typical users include developers, AI architects, and business owners working on contextual AI applications.

What you will accomplish

You will configure a Web Page data source in Gen AI Builder to ingest selected web pages. The content will then be available for transformation and indexing into downstream Libraries and Knowledge Bases.

Why use a Web Page data source

- Web pages often contain valuable, curated, and public-facing information that complements internal data sources.
- Adding relevant web content can enhance AI answers, RAG performance, and semantic search by expanding your Knowledge Base.
- Targeted ingestion of specific web pages ensures the system uses only high-quality and relevant external content.

For guidance on how Knowledge Bases and RAG applications use this content, see:

- Knowledge Bases overview
- Hybrid Manager Gen AI Builder overview

Key considerations

Data relevance and quality

- Ingest web pages that contain useful, high-quality, and relevant content.
- Pages with structured and clearly written HTML are the most effective.
- Avoid dynamic pages that require JavaScript for key content.

Scope of ingestion

- Target specific pages.
- Avoid adding entire websites or root-level URLs unless necessary, as this can result in large volumes of irrelevant data and higher processing costs.

Compliance and terms of service

- Always respect the website's robots.txt file and terms of service.
- Ensure you are authorized to ingest the content for your intended use.

How to configure a Web Page data source

1. In the interface where you manage data sources, select + **Add New Data Source**.
2. Choose **Web Page** from the available data source types.
3. Configure the following fields:
 - **Name**: Provide a clear and unique name. Example: EDB Main Site FAQs .
 - **Description** (optional): Add descriptive context for future reference.
 - **URLs**: Enter one or more full URLs of the web pages to ingest.
 - Example: `https://www.enterprisedb.com/`
 - Select **Add URL** to provide additional URLs.
4. (Optional) Configure advanced options:
 - **Scheduled refresh**: Enable this option to automatically refresh the data on a schedule.
 - Provide a cron expression. Example: `0 2 * * *` (daily at 2 AM).
 - **Transform your data**: Enable this option to apply a PG.AI Structure that transforms the data during ingestion.
 - Select an existing Structure from the list.
5. Select **Create** to add the Web Page data source.

Supported content

- Text content from HTML pages.
- Limited support for client-side rendered content or heavily dynamic pages.

Managing and refreshing the data source

Once created, the Web Page data source can be viewed and managed through the Data Sources interface.

Actions available:

- **Edit**: Modify the data source configuration.
- **Refresh**: Manually trigger a data ingestion job.
- **Delete**: Remove the data source.

You can also review the data job history to monitor ingestion performance and troubleshoot issues.

SDK reference

- Web Scraper Drivers
- Data Loaders

Troubleshooting

URL not accessible

- Check the URL for typos.
- Verify the page is public and not protected by login or access controls.

No or incomplete content

- The page may rely on JavaScript for key content, which is not fully supported.
- Anti-scraping protections may block ingestion.
- Review the data job history. If 0 bytes were ingested, dynamic rendering may be the cause.

Example scenario

You want to enrich your Knowledge Base with relevant articles from the public EDB website.

Example configuration:

- **Name:** EDB Blog - Latest GenAI Articles
- **URL:** <https://www.enterprisedb.com/blog/category/genai>
- **Scheduled refresh:** 0 2 * * * (daily at 2 AM)
- **Transform your data:** Use a Structure to strip unwanted boilerplate such as author bios.

Related topics

- Knowledge Bases overview
- [Create a Knowledge Base in Gen AI Builder](#) <!-- Placeholder link -->
- Hybrid Manager: Using Gen AI Builder

Configure a Custom data source

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users who need to ingest data from proprietary systems, internal databases, or APIs not supported by built-in data source types. Typical users include developers, system integrators, and advanced data engineers who build custom pipelines and Structures.

What you will accomplish

You will configure a Custom data source in Gen AI Builder by selecting a pre-configured PG.AI Structure that acts as a data provider. The output of this Structure will be ingested and made available for indexing into Libraries and Knowledge Bases.

Why use a Custom data source

- Internal data often resides in proprietary databases, APIs, or specialized data stores.
- The Custom data source allows you to connect these systems by writing a PG.AI Structure that performs the data retrieval.
- This offers maximum flexibility while ensuring the resulting content flows through Gen AI Builder's ingestion and transformation pipeline.

For background on how this content powers downstream AI use cases, see:

- Knowledge Bases overview

- *Retrieval Augmented Generation (RAG)*
- *Hybrid Manager: Using Gen AI Builder*

For concepts related to Structures, see:

- *Structures Reference Manual*

Complexity and time to complete

- **Complexity:** High. You must write or have access to a PG.AI Structure that correctly implements data retrieval logic.
- **Estimated time:** 15–60 minutes depending on Structure complexity and target system integration.

Key considerations

Structure design

- The selected PG.AI Structure must be implemented and tested to act as a data source (data provider Structure).
- The Structure should return data in a format suitable for downstream processing (typically textual).
- The Structure should handle retries, error conditions, and paging if applicable.

Transform logic

- You can optionally apply a separate “Transform your data” Structure to further refine, clean, or enrich the content before indexing.

Security and compliance

- If the Structure connects to sensitive systems, ensure appropriate access controls, authentication, and logging are implemented.
- Be mindful of privacy regulations and data handling requirements.

How to configure a Custom data source

1. In the interface where you manage data sources, select + **Add New Data Source**.
2. Choose **Custom Data Source** from the available data source types.
3. Configure the following fields:
 - **Name:** Provide a clear and unique name. Example: `Legacy CRM Connector`.
 - **Description** (optional): Add descriptive context for future reference.
4. Connect to Custom Data Source:
 - **PG.AI Structure:** Select the pre-configured PG.AI Structure that implements your data retrieval logic.
 - Example: `InternalProductAPIReader`
5. (Optional) Configure advanced options:
 - **Scheduled refresh:** Enable this option to automatically refresh the data on a schedule.
 - Provide a cron expression. Example: `0 2 * * *` (daily at 2 AM).
 - **Transform your data:** Enable this option to apply an additional PG.AI Structure to transform the data during ingestion.

- Select an existing Structure from the list.
6. Select **Create** to add the Custom data source.

Supported content

- Determined by the output of the selected PG.AI Structure.
- Typically textual data suitable for AI processing.

Managing and refreshing the data source

Once created, the Custom data source can be viewed and managed through the Data Sources interface.

Actions available:

- **Edit:** Modify the data source configuration.
- **Refresh:** Manually trigger a data ingestion job.
- **Delete:** Remove the data source.

You can also review the data job history to monitor ingestion performance and troubleshoot issues.

SDK reference

- *Structures Reference Manual*
- *Tools Reference Manual*

Troubleshooting

Structure not listed

- Verify that your PG.AI Structure is correctly registered and marked as a data source (data provider Structure).
- Ensure the Structure is accessible in your current project or environment.

Fetching errors or incomplete data

- Check the Structure's implementation for errors, authentication issues, or unexpected API changes.
- Review the data job history for detailed error messages.

Example scenario

You want to ingest product catalog data from an internal REST API into Gen AI Builder.

Example configuration:

- **Name:** Product Catalog via Custom API
- **PG.AI Structure:** `InternalProductAPIReader`
- **Scheduled refresh:** `0 2 * * *` (daily at 2 AM)
- **Transform your data:** Apply a Structure that extracts and formats product descriptions.

Related topics

- Knowledge Bases overview
- *Retrieval Augmented Generation (RAG)*
- *Structures Reference Manual*
- *Embeddings explained*
- *Create a Knowledge Base in Gen AI Builder* <!-- Placeholder link -->
- Hybrid Manager: Using Gen AI Builder

Configure a Data Lake data source

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users who want to ingest file-based content from the platform's managed Data Lake into Gen AI Builder for use in Knowledge Bases and Retrieval-Augmented Generation (RAG) workflows. This is typically used by developers, data engineers, and business owners building AI applications that require internal structured or unstructured documents.

What you will accomplish

You will configure a Data Lake data source in Gen AI Builder to ingest files from a selected Data Lake bucket and (optionally) specific asset paths. The content will then be available for indexing into Libraries and Knowledge Bases.

Why use a Data Lake data source

- Many enterprise documents (PDFs, CSVs, Markdown, text files) already reside in internal Data Lake storage.
- The Data Lake source provides a highly reliable, cost-effective way to bring this content into AI workflows.
- Unlike public web content, Data Lake files are typically higher-trust, internal documents.
- The Data Lake source allows for precise targeting (specific folders or files), making it efficient and scalable.

For background on how this content powers downstream AI use cases, see:

- Knowledge Bases overview
- *Retrieval Augmented Generation (RAG)*
- Hybrid Manager: Using Gen AI Builder

Complexity and time to complete

- **Complexity:** Low to moderate. You need familiarity with your Data Lake structure (bucket names and paths).
- **Estimated time:** 5–10 minutes if paths are known and bucket access is already configured.

Key considerations

Data relevance and quality

- Ingest only trusted, high-quality files that are useful for your AI applications.
- Well-structured files (consistent formatting, well-written text) lead to better AI results.

Scope of ingestion

- Be specific about which files or folders you ingest. Use targeted paths instead of entire buckets where possible.
- Ingesting too broad a scope (e.g., entire raw data buckets) can lead to large processing costs and irrelevant content.

Permissions and access

- You must have appropriate permissions to the selected Data Lake bucket and paths.
- If no specific asset paths are provided, all files in the selected bucket will be ingested.

How to configure a Data Lake data source

1. In the interface where you manage data sources, select + **Add New Data Source**.
2. Choose **Data Lake** from the available data source types.
3. Configure the following fields:
 - **Name**: Provide a clear and unique name. Example: `Q1 Product Specs - Data Lake`.
 - **Description** (optional): Add descriptive context for future reference.
4. Connect to Data Lake:
 - **Bucket**: Select the Data Lake bucket from the dropdown list.
 - **Asset Paths**:
 - Enter one or more specific paths to files or folders. Example: `manuals/pdf/current-version/`
 - Select **Add Asset Path** to add more paths.
 - If no paths are provided, the system will ingest all files in the bucket.
5. (Optional) Configure advanced options:
 - **Scheduled refresh**: Enable this option to automatically refresh the data on a schedule.
 - Provide a cron expression. Example: `0 2 * * *` (daily at 2 AM).
 - **Transform your data**: Enable this option to apply a PG.AI Structure to transform the data during ingestion.
 - Select an existing Structure from the list.
6. Select **Create** to add the Data Lake data source.

Supported file types

- PDF
- CSV
- Markdown
- Most text-based file types

For more about how content is indexed and retrieved from these files, see [Embeddings explained](#).

Managing and refreshing the data source

Once created, the Data Lake data source can be viewed and managed through the Data Sources interface.

Actions available:

- **Edit:** Modify the data source configuration.
- **Refresh:** Manually trigger a data ingestion job.
- **Delete:** Remove the data source.

You can also review the data job history to monitor ingestion performance and troubleshoot issues.

SDK reference

- File Manager Drivers
- Data Loaders

Troubleshooting

Files not found

- Verify that the bucket and asset paths are correct.
- Check the data job history for errors.

Access issues

- Ensure your user or system has the necessary permissions to access the selected bucket and paths.

Example scenario

You want to index a set of PDF product manuals stored in your Data Lake.

Example configuration:

- **Name:** Product Manuals Archive
- **Bucket:** technical-documentation-prd
- **Asset Path:** manuals/pdf/current-version/
- **Scheduled refresh:** 0 2 * * * (daily at 2 AM)
- **Transform your data:** Apply a Structure to extract specific sections from PDFs.

Related topics

- Knowledge Bases overview
- *Embeddings explained*
- *Create a Knowledge Base in Gen AI Builder* <!-- Placeholder link -->
- Hybrid Manager: Using Gen AI Builder

Configure the Data Lake in Gen AI Builder

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users setting up **Gen AI Builder** in **Hybrid Manager AI Factory**. This includes platform administrators, DevOps engineers, and AI builders configuring backend storage for AI Factory pipelines.

What you will accomplish

You will configure a **Data Lake** — the object storage backend required for AI Factory services to function.

You will:

- Create a storage bucket
- Set required CORS policies
- Generate and provide access credentials

Why configure the Data Lake

The Data Lake is required to store and power:

- Uploaded files from Data Sources
- Indexed data and embeddings used by Pipelines and Knowledge Bases
- Structures and Tools required by Assistants
- Temporary artifacts used by AI pipelines

Without a configured Data Lake:

- Libraries, Knowledge Bases, and Assistants will not function.
- RAG pipelines will not operate.
- Data Sources cannot be ingested.

The Data Lake **must be configured before adding Data Sources or creating Knowledge Bases**.

For background:

- [Data Lake Reference Manual](#)
- [Gen AI Builder Concepts \(Assistants, Knowledge, Tools, Threads\)](#)
- [Hybrid Manager: Using Gen AI Builder](#)

Complexity and time to complete

- **Complexity:** Moderate (object storage setup + configuration)
- **Estimated time:** 10–20 minutes

Prerequisites

- Access to an **S3-compatible object storage provider**:
- AWS S3
- Google Cloud Storage (GCS) with S3 interoperability enabled
- Other compatible services (MinIO, Ceph, etc.)
- Permissions to create buckets and manage CORS.
- Ability to obtain object storage credentials.

How to configure the Data Lake

1. Create a dedicated bucket

Provision a new bucket dedicated for Gen AI Builder and Pipelines.

Best practice: Use a dedicated bucket — do not reuse general-purpose or mixed-content buckets.

AWS S3 or compatible

- Create a new bucket (example: `genai-data-lake-hcp`).

Google Cloud Storage (GCS)

- Create a bucket.
- Enable **S3 interoperability**.
- Assign required roles:
 - Storage Admin
 - Service Account Token Creator
- Generate HMAC keys (used as Access Key ID and Secret Access Key).

2. Configure CORS policy

You must configure **Cross-Origin Resource Sharing (CORS)** to allow Gen AI Builder console and APIs to interact with the Data Lake.

Example CORS for S3-compatible storage

```
[
  {
    "AllowedHeaders": [ "*" ],
    "AllowedMethods": [ "PUT", "POST", "DELETE", "GET", "HEAD" ],
    "AllowedOrigins": [ "https://<PORTAL_DOMAIN_NAME>" ],
    "ExposeHeaders": [ ]
  }
]
```

Example CORS for GCS

1. Create `cors-config.json`:

```
[
  {
    "origin": ["https://<PORTAL_DOMAIN_NAME>"],
    "method": ["GET", "PUT", "POST", "DELETE", "HEAD"],
    "responseHeader": ["*"],
    "maxAgeSeconds": 3600
  }
]
```

2. Apply CORS policy:

```
gsutil cors set cors-config.json gs://<your-gcs-bucket-name>
```

3. Obtain storage credentials

You will need:

- Bucket name
- S3 Endpoint URL
- Access Key ID
- Secret Access Key
- Region (if applicable)

For GCS: use HMAC keys as your Access Key ID and Secret Access Key.

4. Provide credentials to your deployment

Configure your Kubernetes deployment or backend:

- Use environment variables or a configuration file.
- Apply as part of your Helm chart, secrets manager, or Kubernetes Secret.

Example environment variables:

```
DATA_LAKE_BUCKET_NAME=genai-data-lake-hcp
DATA_LAKE_ENDPOINT_URL=https://<your-endpoint>
DATA_LAKE_ACCESS_KEY_ID=<access-key-id>
DATA_LAKE_SECRET_ACCESS_KEY=<secret-access-key>
DATA_LAKE_REGION=<region> # If applicable
```

5. Verify connectivity

1. Log into Gen AI Builder → **Data Lake** section.
2. Confirm your configured bucket appears.
3. Create a folder or upload a test file to verify permissions.

Example: `genai-data-lake-hcp` → Create Folder → Upload File.

Troubleshooting

Bucket not visible in Gen AI Builder

- Verify that credentials are correct.
- Check that your CORS policy allows the required HTTP methods.
- Ensure that the bucket is dedicated to Gen AI Builder / AI Factory.

Uploads or console interactions fail

- Confirm CORS settings.
- Ensure that API keys or credentials have correct permissions.

Access denied errors

- Confirm the Access Key ID and Secret Access Key have required permissions:
- S3 or Storage Object Admin
- Read / write / delete permissions on the bucket

SDK reference

- Data Artifacts — pass data between components
- File Manager Drivers — object storage access patterns

Related topics

- *Data Lake Reference Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- *Configure Data Sources in Gen AI Builder*
- *Knowledge Bases Reference Manual*
- Hybrid Manager: Using Gen AI Builder

Create a Retriever in Gen AI Builder

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users configuring **Retrievers** in **Gen AI Builder** in **Hybrid Manager AI Factory**. Typical users include developers, AI architects, and conversational designers.

What you will accomplish

You will create a **Retriever** to control **which Knowledge Bases** your Assistants query and **how** content is retrieved.

You will configure:

- Target Knowledge Bases
- Max Tokens limit
- Optional metadata

Retrievers enable **grounded, context-aware** AI responses for your Assistants.

Why create a Retriever

Retrievers are a key component of the **Retrieval-Augmented Generation (RAG)** pipeline:

- Define which **Knowledge Bases** Assistants query.
- Control **semantic search** and **context size** passed to the LLM.
- Enable metadata injection for auditing, versioning, or custom logic.
- Without a Retriever, an Assistant cannot retrieve content to ground its responses.

For background, see:

- [*Retrievers Reference Manual*](#)
- [*Knowledge Bases Reference Manual*](#)
- [*Retrieval Augmented Generation \(RAG\)*](#)

To see how Retrievers are used in **Hybrid Manager deployments**, visit:

- Hybrid Manager: Using Gen AI Builder

Complexity and time to complete

- **Complexity:** Low to moderate
- **Estimated time:** 5–10 minutes

When to create a Retriever

- Before assigning **RAG capabilities** to an Assistant.
- When tuning how an Assistant retrieves and prioritizes knowledge.
- When combining content from **multiple Knowledge Bases**.
- When customizing retrieval behavior for different personas or use cases.

How to create a Retriever

1. Navigate to Retrievers

- In Gen AI Builder console, go to **Retrievers**.

2. Create a new Retriever

- Click **Create Retriever**.

3. Configure Retriever fields

Name (required)

- Provide a clear, unique name.
- Example: `Product Docs Retriever`

Description (optional)

- Describe the Retriever's purpose or intended use.
- Example: `Retriever for product documentation and customer FAQs.`

Knowledge Bases (required)

- Select one or more existing **Knowledge Bases** to target.
- Example: `PG Financial Product FAQs KB, PG Financial Account Services KB`

Max Tokens (optional)

- Set the **Max Tokens** limit for context passed to the LLM.
- Example: `2000`

Guidance:

- Higher token limits → more context but larger LLM prompts.
- Lower token limits → faster response but may miss needed content.
- Typical range: **1000–3000**.

Metadata (optional)

- Provide optional metadata as a valid JSON object.
- Example: `{"version": "1.0", "team": "AI Engineering"}`

Usage:

- Tag or version your Retriever configurations.
- Drive advanced logic in custom Griptape Structures or Assistant templates.

4. Finalize creation

- Click **Create**.
- The new Retriever will appear in the **Retrievers** list.
- It is now available to assign to one or more Assistants.

Associating a Retriever with an Assistant

- When configuring an **Assistant**, go to **Advanced Options**.
- Select one or more Retrievers to associate.
- This controls **which Knowledge Bases** the Assistant queries at runtime.

Tip: Use **different Retrievers** for different personas or Assistant use cases — enabling fine-grained RAG control.

Example scenario

You want to create a Retriever for an Assistant that answers customer product questions:

- Name: `Product Support Retriever`
- Description: `Fetches product specs and pricing from KBs.`
- Knowledge Bases:
 - `Product Catalog Hybrid KB`
 - `Pricing Policies KB`
- Max Tokens: `2000`
- Metadata: `{"team": "Support AI"}`

SDK reference

- RAG Engines — retrieval modules and response assembly
- Embedding Drivers
- Rerank Drivers
- Vector Store Drivers

Troubleshooting

Retriever not returning expected results

- Verify that the target **Knowledge Bases** contain relevant content.
- Check your **Max Tokens** setting.
- Confirm that the correct Retriever is assigned to the Assistant.

Retriever not visible when configuring an Assistant

- Ensure the Retriever was created successfully.
- Verify that the selected Knowledge Bases are accessible and indexed.

Token impact

- Setting too high may exceed LLM context window → risk truncation.
- Setting too low may result in incomplete or unhelpful responses.

Tip: Start with **2000 tokens** and adjust based on Assistant testing.

Related topics

- *Retrievers Reference Manual*
- *Knowledge Bases Reference Manual*
- *Retrieval Augmented Generation (RAG)*
- *Structures Reference Manual*
- Hybrid Manager: Using Gen AI Builder

How to build assistants

Create a Ruleset in Gen AI Builder

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users building **Assistants** or **Structures** in **Gen AI Builder** in **Hybrid Manager AI Factory**. Typical users include developers, AI architects, content designers, and compliance owners.

What you will accomplish

You will create a **Ruleset** — a collection of **Rules** written in natural language to guide the behavior, tone, and compliance of your Assistants or Structures.

You will:

- Define one or more Rules.
- Package them into a Ruleset.
- Make the Ruleset available for use in Assistants.

Why create a Ruleset

- Rulesets define **behavioral guidelines** and **governance** for AI-driven interactions.
- They control tone, style, and compliance in responses.
- They help implement different **Assistant personas**.
- They enable consistent behavior across Assistants and applications.
- They complement Knowledge Bases and Retrievers in the AI pipeline.

For background, see:

- *Rulesets Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- Hybrid Manager: Using Gen AI Builder

Complexity and time to complete

- **Complexity:** Low
- **Estimated time:** 5–10 minutes

When to create a Ruleset

- When defining the **tone**, **voice**, or **policy constraints** for an Assistant.
- When implementing Assistants for different personas or user audiences.
- When ensuring compliance with **brand standards** or **regulatory guidelines**.
- Before assigning an Assistant to a production environment.

Pre-requisites

- Gen AI Builder deployed with **Assistant features** enabled.

How to create a Ruleset

1. Navigate to Rulesets

- In Gen AI Builder console, go to **Rulesets**.

2. Create a new Ruleset

- Click **Create Ruleset**.

3. Configure Ruleset fields

Name (required)

- Provide a clear, unique name.
- **Example:** `Customer Service Tone Rules`

Description (optional)

- Describe the Ruleset's purpose or scope.
- **Example:** `Guidelines for polite, professional responses in customer interactions.`

Rules (required)

- Select one or more existing **Rules** to populate the Ruleset.
- If needed, create new Rules during this step.

Creating a new Rule

- In the **Rules** field, click **Create new rule**.
- In the modal:
 - **Name:** `Maintain Polite Tone`
 - **Rule text:** `Always address the user respectfully. Use polite greetings and sign-offs.`

- Click **Create** to save the Rule.
- Repeat to add additional Rules.

Alias (optional)

- Provide an optional **alias** — a user-friendly reference name.
- Example: `polite-support`

Metadata (optional)

- Provide optional metadata as a valid JSON object.
- Example: `{ "version": "1.0", "author": "support_team" }`

4. Finalize Ruleset creation

- Click **Create** to save the Ruleset.
- The new Ruleset will appear in the **Rulesets** list.
- It is now available to assign to Assistants.

Associating a Ruleset with an Assistant

- When configuring an **Assistant**, go to **Behavior and Rules** or **Advanced Options**.
- Select one or more **Rulesets** to assign.
- The Ruleset will be used as **behavioral guidance** during Assistant interactions.

Tip: Use modular Rulesets — combine **global**, **persona-specific**, and **compliance** Rulesets for layered behavior control.

Example scenario

You want to create a Ruleset to enforce a polite, professional tone for a **Customer Support Assistant**.

Example configuration:

- **Name:** `Polite Support Tone`
- **Description:** `Ensures the Assistant maintains a polite and professional tone during customer interactions.`
- **Alias:** `polite-support`
- **Metadata:** `{ "version": "1.0", "author": "support_team" }`

Rules:

1. **Use Formal Salutations** Always begin responses with a polite greeting, such as "Hello," or "Good day,". Address the user respectfully if their name is known.
2. **Offer Assistance Clearly** Clearly state how you can assist the user. For example, "How may I help you today?" or "I can assist you with..."

After creation:

- Assign the Ruleset to your **Customer Support Assistant**.

Troubleshooting

Ruleset not visible when configuring an Assistant

- Ensure the Ruleset has at least one Rule.
- Confirm the Ruleset was created successfully.

Conflicting or ambiguous Rules

- Review Rule text carefully — avoid **overlapping** or **contradictory** instructions.
- Test Assistants with the Ruleset applied and refine as needed.

Ruleset updates not reflected

- Verify that the **Assistant configuration** references the latest Ruleset version.
- Re-assign or refresh the Assistant if necessary.

Best practices

- Use **modular** Rulesets — separate compliance, tone, and persona guidelines.
- Keep **Rules clear and unambiguous** — avoid conflicting instructions.
- Implement **versioning** via Ruleset metadata.
- Review Assistant responses regularly to validate behavior against assigned Rulesets.

Related topics

- *Rulesets Manual*
- *Structures Reference Manual*
- *Assistants Reference Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- Hybrid Manager: Using Gen AI Builder

Create a Structure in Gen AI Builder

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Who is this for

Platform users building advanced AI workflows in **Gen AI Builder** in **Hybrid Manager AI Factory**. Typical users include developers, AI architects, and data engineers.

What you will accomplish

You will create a **Structure** in Gen AI Builder by deploying an agent, pipeline, or workflow packaged as a Zip file. You will configure your Structure for execution in AI Factory.

Why create a Structure

- Structures power **advanced AI workflows** beyond basic conversation.
- They enable:
 - **Data transformation** for Data Sources.
 - **Task execution** via Tools in Assistants.
 - **Custom Retrieval pipelines**.
 - **Integration with external APIs** and services.
- Reusable business logic components across AI Factory applications.

For background, see:

- *Structures Reference Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- Hybrid Manager: Using Gen AI Builder

Complexity and time to complete

- **Complexity**: Moderate
- **Estimated time**: 10–20 minutes

When to create a Structure

- When building **custom Tools** callable by Assistants.
- When transforming or enriching data in Data Source pipelines.
- When implementing complex **AI pipelines** that require multi-step logic.
- When integrating external **APIs**, **systems**, or **business workflows**.

Pre-requisites

- Structure packaged as a **Zip file**:
- Includes Structure code and dependencies.
- Contains Structure configuration (typically YAML).
- Uploaded to your configured **Data Lake**.

How to create a Structure

1. Navigate to Structures

- In Gen AI Builder console, go to **Structures**.

2. Create a new Structure

- Click **Create Structure**.

3. Select creation method

- Choose **Create Structure from Zip file**.

4. Configure Structure fields

Name (required)

- Provide a clear, unique name.
- Example: `PGFT_TransactionCategorizer`

Description (optional)

- Describe what the Structure does.
- Example: `Categorizes financial transactions using PGFT business logic.`

5. Configure Data Lake parameters

Bucket Id (required)

- Select the **Data Lake bucket** where your Structure Zip file is stored.

Asset Path (required)

- Provide the full path to your Zip file inside the selected bucket.
- Example: `griptape-structures/transaction_categorizer.zip`

6. Configure Structure Config parameters

Structure Config File (recommended)

- Enter the relative path inside the Zip to your Structure Config file.
- Example: `structure_config.yaml`

7. Configure environment variables (optional)

You can define environment variables that will be available to the Structure.

Add individual variables

- **Type:** Variable or Secret.
- **Name:** Example: `API_KEY`
- **Value:** Example: `your_actual_api_key_here`

Import .env file

- Optionally paste contents of a `.env` file to bulk import variables.

8. Finalize creation

- Click **Create**.

The system will:

- Unzip your file.
- Set up the runtime environment.
- Make the Structure available in AI Factory.

9. Verify deployment

- Your Structure will appear in the **Structures** list.
- Monitor **deployment status**.
- You can now:
 - Run the Structure manually via the console or API.
 - Use the Structure as a **Tool** in Assistants.
 - Use the Structure in **Data Source pipelines**.

Example scenario: Deploy a Transaction Categorizer Structure

1. Package Griptape Structure `TransactionCategorizer`.
2. Upload to Data Lake:
 - Bucket: `pgai-structures`
 - Path: `transaction_categorizer/transaction_categorizer.zip`
3. Create Structure:
 - Name: `PGFT_TransactionCategorizer`
 - Description: `Categorizes financial transactions.`
 - Bucket Id: `pgai-structures`
 - Asset Path: `transaction_categorizer/transaction_categorizer.zip`
 - Structure Config File: `structure_config.yaml`
4. Add env variables:
 - Name: `MODEL_API_KEY`
 - Value: `your_key_here`
5. Click **Create**.

Next steps:

- Run the Structure to verify behavior.
- Assign the Structure as a **Tool** to Assistants.
- Use the Structure in Data Source pipelines if applicable.

SDK reference

- *Structures Reference Manual*
- *Structures Reference Manual*
- *Tools Reference Manual*
- RAG Engines

Troubleshooting

Structure fails to deploy

- Verify Zip file contents:
- Structure code is valid.
- Structure Config file is present and correctly referenced.
- Confirm the **Data Lake path** is correct and accessible.

Environment variables not working

- Confirm variables are entered correctly.
- If using Secrets, ensure Secrets are properly configured.

Structure not appearing as a Tool

- Verify the Structure implements a **Tool-compatible interface**.
- Confirm the Structure is published and available for Tool assignment.

Structure errors at runtime

- Review Structure logs in Gen AI Builder console.
- Verify external API endpoints and credentials.
- Check Structure Config settings.

Best practices

- Use modular, reusable Structure design.
- Externalize configuration via environment variables.
- Implement **robust error handling**.
- Monitor execution performance — validate latency and throughput.
- Version your Structure artifacts in Data Lake.

Related topics

- *Structures Reference Manual*
- *Structures Reference Manual*
- *Create an Assistant in Gen AI Builder*
- *Rulesets Manual*

- *Knowledge Bases Reference Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- Hybrid Manager: Using Gen AI Builder

Create a Tool in Gen AI Builder

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

Create a Tool in Gen AI Builder

This guide demonstrates deploying custom tools that enable AI assistants to interact with external systems, perform calculations, and execute business logic within Gen AI Builder.

Prerequisites

Required Components

- Tool packaged as ZIP file containing implementation code and dependencies
- Tool configuration file (YAML or Python format)
- ZIP file uploaded to configured Data Lake bucket
- Gen AI Builder access with appropriate permissions

Development Dependencies

- Tool implementation using [pgai-ai-builder SDK](#)
- External API credentials or database access (if applicable)
- Testing environment for tool validation

Implementation Steps

1. Access Tool Management

Navigate to the **Tools** section within Gen AI Builder console.

2. Initialize Tool Creation

1. Click **Create Tool** to begin the deployment process
2. Select **Griptape Tool from Data Lake** as the creation method

3. Configure Tool Identity

Tool Name (Required)

- Provide descriptive, unique identifier
- Use clear naming convention: `Organization_Category_Function`
- Example: `PGAI_Financial_StockTicker`

Description (Optional)

- Document tool functionality and use cases
- Include integration details and expected behavior
- Example: Retrieves real-time stock prices from external financial API

4. Configure Data Lake Integration

Bucket Selection (Required)

- Select Data Lake bucket containing your tool package
- Verify bucket accessibility and permissions

Asset Path (Required)

- Specify complete path to ZIP file within selected bucket
- Path format: `category/tool_name/package.zip`
- Example: `financial_tools/stock_ticker/stock_ticker_v1.2.zip`

5. Configure Tool Parameters

Tool Configuration File (Recommended)

- Specify relative path within ZIP to configuration file
- Standard configuration files: `tool_config.yaml` or `config.py`
- Configuration defines tool parameters, validation rules, and execution settings

Example Configuration Structure:

```
# tool_config.yaml

name: "stock_ticker_tool"
version: "1.2.0"
description: "Retrieves stock market data"
parameters:
  - name: "symbol"
    type: "string"
    required: true
    description: "Stock ticker symbol"
  - name: "period"
    type: "string"
    default: "1d"
    options: ["1d", "5d", "1mo"]
```

6. Environment Configuration

Environment Variables Configure runtime environment for tool execution:

Individual Variable Configuration

- **Type:** Select Variable (plain text) or Secret (encrypted storage)
- **Name:** Environment variable identifier
- **Value:** Configuration value or credential

Common Variable Examples:

```
API_KEY = your_api_key_here
BASE_URL = https://api.example.com
TIMEOUT_SECONDS = 30
RETRY_ATTEMPTS = 3
```

Bulk Import Option

- Import multiple variables using .env file format
- Paste complete .env file contents for batch configuration

7. Deploy Tool

Click **Create** to initiate deployment process.

Deployment Process:

1. System validates ZIP file structure and contents
2. Tool configuration parsing and validation
3. Runtime environment preparation
4. Tool registration in Gen AI Builder catalog

8. Verify Deployment

Post-Deployment Validation:

- Tool appears in Tools catalog with Active status
- Configuration parameters display correctly
- Environment variables configured appropriately
- Tool available for assistant assignment

Practical Example: Financial Exchange Rate Tool

Tool Package Structure

```
fx_rate_tool.zip
├── fx_rate_tool.py           # Tool implementation
├── requirements.txt         # Python dependencies
├── tool_config.yaml         # Tool configuration
└── README.md               # Documentation
```

Deployment Configuration

Basic Information:

- **Name:** PGAI_Financial_FXRate
- **Description:** Retrieves current foreign exchange rates from external API

Data Lake Configuration:

- **Bucket:** pgai-financial-tools
- **Asset Path:** fx_tools/fx_rate_tool_v2.1.zip

- **Config File:** tool_config.yaml

Environment Variables:

```
FX_API_KEY = abcd1234-your-api-key-here
FX_API_URL = https://api.exchangerate-api.com/v4/latest
CACHE_DURATION = 300
MAX_RETRIES = 3
```

Tool Configuration File

```
name: "fx_rate_tool"
version: "2.1.0"
description: "Foreign exchange rate retrieval tool"
parameters:
  - name: "base_currency"
    type: "string"
    required: true
    description: "Base currency code (USD, EUR, etc.)"
  - name: "target_currency"
    type: "string"
    required: true
    description: "Target currency for conversion"
  - name: "amount"
    type: "number"
    default: 1.0
    description: "Amount to convert"
```

Troubleshooting

Deployment Issues

ZIP File Validation Errors

- Verify ZIP file contains required implementation files
- Confirm tool configuration file path accuracy
- Validate Python dependency specifications in requirements.txt

Data Lake Access Problems

- Verify bucket selection and permissions
- Confirm asset path accuracy including file extension
- Check Data Lake connectivity and authentication

Configuration Issues

Environment Variable Problems

- Verify variable names match tool implementation expectations
- Confirm Secret-type variables for sensitive credentials
- Validate variable values and formatting

Tool Configuration Errors

- Review configuration file syntax (YAML formatting)
- Verify parameter definitions match tool implementation
- Check required parameter specifications

Runtime Issues

Tool Execution Failures

- Review tool logs in Gen AI Builder console
- Validate external API endpoints and credentials
- Verify network connectivity for external integrations
- Check tool implementation error handling

Assistant Integration Problems

- Confirm tool status shows as Active in catalog
- Verify tool permissions and availability
- Check assistant configuration for tool assignment

Best Practices

Tool Development

Implementation Guidelines

- Develop tools as single-purpose, modular components
- Implement comprehensive error handling and timeout management
- Use pgai-ai-builder SDK for consistent interfaces and utilities
- Include detailed logging for debugging and monitoring

Configuration Management

- Externalize all credentials and API keys through environment variables
- Use descriptive parameter names and comprehensive documentation
- Implement parameter validation and default value handling
- Version tool packages systematically for deployment tracking

Security Considerations

Credential Management

- Store sensitive information using Secret-type environment variables
- Rotate API keys and credentials regularly
- Implement least-privilege access for external system integration
- Monitor tool usage for unusual activity patterns

Access Control

- Limit tool availability to appropriate user groups
- Implement audit logging for tool execution tracking

- Validate input parameters to prevent injection attacks
- Use secure communication protocols for external API calls

Operational Excellence

Testing Procedures

- Test tools independently before assistant integration
- Validate functionality across expected input ranges
- Verify error handling under various failure conditions
- Monitor performance characteristics and resource utilization

Monitoring and Maintenance

- Monitor tool execution logs and error rates
- Track performance metrics and usage patterns
- Plan regular updates for dependency management
- Maintain version control for tool artifacts in Data Lake

SDK reference

- *Tools Reference Manual*
- *Structures Reference Manual*

Next Steps

After successful tool deployment:

1. **Test Tool Functionality**
 - Use tool testing capabilities within Gen AI Builder
 - Validate parameter handling and response formatting
 - Verify error handling under various conditions
2. **Integrate with Assistants**
 - Assign tool to appropriate assistants based on use cases
 - Configure tool usage policies through rulesets
 - Test assistant interactions with tool capabilities
3. **Monitor Performance**
 - Track tool execution metrics and success rates
 - Monitor resource utilization and response times
 - Establish alerting for tool failures or performance degradation

Related Documentation

Implementation Resources

- [pgai-ai-builder SDK](#) : Complete development framework
- [Tool Configuration Reference](#) : Detailed configuration options
- [Tool Execution Testing](#) : Testing procedures and validation methods

Integration Guides

- [Tools Reference Manual](#) : Comprehensive tool architecture and capabilities
- [Create an Assistant in Gen AI Builder](#) : Tool assignment procedures
- [Structures Reference Manual](#) : Tool integration in automated processes

Create an Assistant in Gen AI Builder

Who is this for

Platform users building AI-driven Assistants in **Gen AI Builder** in **Hybrid Manager AI Factory**. Typical users include developers, AI architects, conversational designers, and content owners.

What you will accomplish

You will create an **Assistant** — an AI-powered agent that can:

- Interact with users through natural language
- Retrieve knowledge from Knowledge Bases
- Follow behavioral guidelines using Rulesets
- Optionally perform actions via Tools

You will configure the core components of an Assistant and prepare it for use in production or testing.

Why create an Assistant

Assistants provide the **interactive layer** for your AI Factory content and capabilities — enabling trusted, conversational access to your organization’s knowledge and services.

They unify:

- Large Language Models (LLMs)
- Knowledge Bases
- Rulesets
- Tools
- Memory (for context-aware interactions)

Assistants allow end users to interact with your AI in a **guided, governed, and useful way** — powering applications such as internal copilots, support agents, and knowledge assistants.

For background, see:

- [Assistants Reference Manual](#)
- [Gen AI Builder Concepts \(Assistants, Knowledge, Tools, Threads\)](#)
- [Hybrid Manager: Using Gen AI Builder](#)

Complexity and time to complete

- **Complexity:** Moderate
- **Estimated time:** 10–15 minutes

When to create an Assistant

- When you want to expose Knowledge Bases through a conversational interface.
- When you want to control AI behavior and tone through Rulesets.
- When you need the AI to perform external tasks via Tools.
- As the first step in building an AI-powered experience (copilot, chatbot, API agent, etc.).

Pre-requisites

- At least one Knowledge Base (optional but strongly recommended).
- One or more Rulesets (optional but strongly recommended).
- (Optional) Tools configured and available.

How to create an Assistant

1. Navigate to Assistants

- In Gen AI Builder console, go to **Assistants**.
- This is typically found under **APPS** or **Agent Studio**.

2. Create a new Assistant

- Click **Create Assistant**.

3. Configure Assistant fields

Name (required)

- Provide a unique, descriptive name.
- Example: PG Financial Client Support AI

Description (optional)

- Describe the Assistant's purpose and target audience.
- Example: Virtual support agent for PG Financial clients.

LLM Model Selection (required)

- Select the Large Language Model (LLM) to power the Assistant.
- Example options:
 - GPT-4
 - Claude 3 Sonnet
 - Gemini Pro

- Custom models deployed via *Model Serving Reference Manual*

System Prompt / Instructions (optional)

- Define the Assistant's persona or high-level behavior.
- Example: You are a helpful and friendly customer support agent for PG Financial.

Knowledge Bases (optional)

- Select one or more existing Knowledge Bases to enable **Retrieval-Augmented Generation (RAG)**.
- Example: PG Financial Product FAQs KB, PG Financial Account Services KB

Rulesets (optional)

- Select one or more existing Rulesets to define behavioral guidelines.
- Example: Polite Support Tone, Financial Advice Disclaimer

Tools (optional)

- Select Tools the Assistant can invoke (if Tools are configured).
- Example Tools:
- Calculator
- Calendar lookup
- Custom API connector

SDK reference

- *Structures Reference Manual*
- Assistant Drivers
- Conversation Memory Drivers
- *Tools Reference Manual*

Memory Configuration (optional, advanced)

- Configure how the Assistant remembers conversation context.
- Typical options:
- Number of turns to remember
- Summarization or summarization + truncation strategy

Other Advanced Settings (optional)

- Temperature (controls response creativity)
- Top_p (controls nucleus sampling)
- Tool usage settings
- RAG behavior settings (advanced tuning)

4. Finalize creation

- Click **Create** to save the Assistant.
- The new Assistant will appear in the **Assistants** list.

Example: Create a Support Assistant

1. Navigate to **Assistants**.
2. Click **Create Assistant**.
3. Configure fields:
 - **Name:** PG Financial Client Support AI
 - **Description:** Virtual agent providing customer support for PG Financial.
 - **LLM Model:** GPT-4
 - **System Prompt:** You are a polite and helpful customer support agent for PG Financial.
 - **Knowledge Bases:**
 - PG Financial Product FAQs KB
 - PG Financial Account Services KB
 - **Rulesets:**
 - Polite Support Tone
 - Financial Advice Disclaimer
 - **Tools:**
 - Account Balance Lookup API (example)
4. Click **Create**.
5. Next steps:
 - Test the Assistant using the built-in test chat UI.
 - Refine Rulesets, Tools, and Memory settings based on results.

Troubleshooting

Assistant not responding as expected

- Verify that:
 - Knowledge Bases contain relevant content.
 - Appropriate Rulesets are assigned.
 - System Prompt aligns with desired behavior.

RAG behavior inconsistent

- Check Retriever configuration and Knowledge Base indexing.
- Verify that Tools are configured correctly if used.

Memory not working as expected

- Review Memory settings.
- Adjust summarization parameters if applicable.

Tools not functioning

- Verify Tool configuration and API availability.
- Confirm that the Assistant has the Tool assigned.

Related topics

- *Assistants Reference Manual*
- *Rulesets Manual*
- *Knowledge Bases Reference Manual*
- *Retrievers Reference Manual*
- *Structures Reference Manual*
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- Hybrid Manager: Using Gen AI Builder

How to integrate with your apps

Customer Service Assistant Implementation Guide

This implementation guide demonstrates building a governed customer service assistant using Hybrid Manager's integrated AI capabilities. The assistant retrieves accurate information from organizational knowledge bases and generates responses using models deployed within your controlled environment.

Implementation Outcome: Production-ready assistant in Agent Studio with comprehensive knowledge retrieval, governed response generation, and full operational visibility.

SDK reference

- *Structures Reference Manual*
- *Tools Reference Manual*
- RAG Engines

Prerequisites and Architecture Requirements

Infrastructure Dependencies

- Hybrid Manager cluster with AI Factory capabilities enabled
- Sufficient compute resources for model serving and embedding generation
- Network connectivity to organizational data sources
- Appropriate permissions for Gen AI Builder and Agent Studio operations

Access Control Requirements

- Gen AI Builder permissions for knowledge base creation and management
- Agent Studio access for assistant configuration and testing
- Model Serving permissions for deploying or accessing inference endpoints
- Data source access aligned with organizational security policies

Data Preparation

Prepare customer service content including documentation, FAQ databases, policy documents, and procedural guides. Content should be structured for optimal retrieval performance with clear source attribution.

Recommended Content Types:

- Customer support documentation with clear section hierarchies
 - FAQ databases with question-answer pairs
 - Policy documents with structured procedures
 - Troubleshooting guides with step-by-step instructions
-

Implementation Workflow

Phase 1: Knowledge Base Configuration

Knowledge bases provide the factual foundation for assistant responses through semantic search across organizational content.

Data Source Integration

1. Content Assessment

- Evaluate existing customer service documentation for completeness and accuracy
- Identify gaps in current content coverage
- Establish content update procedures for maintaining knowledge freshness

2. Processing Configuration

- Configure appropriate chunking strategies based on document structure
- Implement metadata extraction for source attribution
- Establish embedding generation using organizational models

3. Quality Validation

- Test retrieval accuracy using representative customer queries
- Validate source attribution and citation generation
- Establish performance baselines for retrieval latency

Implementation Resources:

- *Create a Knowledge Base in Gen AI Builder*
- Pipeline Knowledge Base Configuration

Auto-Processing Implementation

Configure automated content updates to maintain knowledge base accuracy without manual intervention.

```
# Example auto-processing configuration

processing_schedule: "0 2 * * *" # Daily at 2 AM
content_sources:
  - type: "object_storage"
    path: "s3://customer-docs/support/"
    include_patterns: ["*.pdf", "*.docx", "*.md"]
update_strategy: "incremental"
embedding_refresh: "on_change"
```

Critical Considerations:

- Balance update frequency with computational resource consumption
- Implement change detection to avoid unnecessary processing
- Monitor embedding drift over time with content updates

Phase 2: Assistant Behavior Configuration

Ruleset Development

Rulesets constrain assistant behavior to ensure responses align with organizational standards and compliance requirements.

Behavioral Guidelines:

- Professional tone appropriate for customer interactions
- Accurate information delivery with appropriate disclaimers
- Escalation procedures for complex queries beyond assistant capabilities
- Data privacy controls for sensitive customer information

Example Ruleset Structure:

```
# Customer Service Assistant Guidelines

## Response Standards

- Provide accurate, helpful information based solely on knowledge base content
- Maintain professional, empathetic tone in all interactions
- Include source citations for all factual claims
- Acknowledge limitations when information is unavailable

## Escalation Criteria

- Complex technical issues requiring human expertise
```

(continues on next page)

(continued from previous page)

```

- Account-specific information requiring authentication
- Complaints or sensitive customer concerns
- Requests for policy exceptions or special handling

## Compliance Requirements

- Never request or process personally identifiable information
- Follow data retention policies for conversation logs
- Maintain audit trails for all customer interactions

```

Retrieval Strategy Configuration

Configure retrieval parameters to optimize accuracy and relevance for customer service queries.

Key Configuration Areas:

1. Similarity Thresholds

- Set minimum relevance scores to prevent low-quality matches
- Balance recall (finding relevant information) with precision (avoiding noise)
- Establish different thresholds for different content types

2. Result Limits

- Configure top-K values based on response generation requirements
- Consider computational overhead for large result sets
- Implement dynamic adjustment based on query complexity

3. Content Filtering

- Apply metadata-based filters for content recency
- Implement access control filters based on user permissions
- Configure content type preferences for different query categories

Retrieval Configuration Example:

```

{
  "similarity_threshold": 0.75,
  "max_results": 8,
  "content_filters": {
    "recency_days": 365,
    "content_types": ["faq", "documentation", "policy"],
    "access_level": "customer_facing"
  },
  "reranking": {
    "enabled": true,
    "model": "organizational_rerank_model"
  }
}

```

(continues on next page)

(continued from previous page)

```
}
}
```

Phase 3: Model Integration

Model Selection and Deployment

Choose appropriate language models based on customer service requirements including response quality, latency, and operational costs.

Model Characteristics for Customer Service:

- Appropriate response length for customer queries
- Professional tone generation capabilities
- Accurate information synthesis from retrieved context
- Consistent performance under varying load conditions

Deployment Considerations:

- Resource allocation for expected concurrent conversations
- Auto-scaling configuration for peak support periods
- Health monitoring for model availability and performance
- Fallback procedures for model unavailability

External Provider Integration

When organizational policies permit external model usage, configure appropriate access controls and monitoring.

Security Requirements:

- API key management aligned with organizational security policies
- Request/response logging for audit and troubleshooting
- Data handling policies for information sent to external providers
- Cost monitoring and usage controls

Phase 4: Assistant Assembly and Testing

Component Integration

Assemble knowledge bases, rulesets, retrievers, and models into functional assistant configurations within Agent Studio.

Integration Checklist:

- ☐ Knowledge base connectivity and retrieval validation
- ☐ Ruleset application and behavior verification
- ☐ Model endpoint accessibility and response generation
- ☐ Tool integration for external system connectivity (if applicable)
- ☐ Citation and source attribution accuracy

Comprehensive Testing Protocol

Systematic testing ensures assistant reliability before production deployment.

Testing Categories:

1. Functional Validation

- Query processing across diverse customer service scenarios
- Response accuracy compared to ground truth documentation
- Citation verification for all factual claims
- Error handling for queries outside knowledge base scope

2. Performance Testing

- Response latency under normal and peak load conditions
- Concurrent user handling capabilities
- Resource utilization monitoring during operation
- Scaling behavior validation

3. Behavioral Compliance

- Ruleset adherence across conversation scenarios
- Appropriate escalation triggering
- Consistent tone and professionalism
- Data privacy policy compliance

Testing Implementation:

```
# Example testing framework structure

class AssistantTestSuite:
    def test_response_accuracy(self):
        """Validate responses against known correct answers."""
        test_queries = [
            "What is the return policy for electronics?",
            "How do I reset my account password?",
            "What are the shipping options available?"
        ]

        for query in test_queries:
            response = self.assistant.query(query)
            assert self.validate_accuracy(response)
            assert self.validate_citations(response)

    def test_performance_characteristics(self):
        """Measure response times and resource usage."""
        # Implementation details for load testing

    def test_behavioral_compliance(self):
        """Verify adherence to organizational guidelines."""
        # Implementation details for compliance testing
```

Configuration Optimization

Performance Tuning Parameters

Retrieval Configuration:

- `top_k`: Start with 5-10 results, adjust based on response quality
- `similarity_threshold`: Begin at 0.7, increase to reduce noise or decrease to improve recall
- `context_window`: Balance comprehensive context with generation speed

Model Parameters:

- `temperature`: Use 0.1-0.3 for consistent, factual responses
- `max_tokens`: Configure based on typical response length requirements
- `timeout`: Set appropriate values for customer experience expectations

Auto-processing Settings:

- Update frequency aligned with content change patterns
- Resource allocation for processing operations
- Monitoring thresholds for processing failures

Operational Monitoring

Implement comprehensive monitoring for production assistant operations.

Key Metrics:

- Response accuracy rates through user feedback
- Query resolution rates without escalation
- Average response latency and 95th percentile measurements
- Knowledge base hit rates and retrieval effectiveness
- Model performance and availability statistics

Alerting Configuration:

- Response latency exceeding service level objectives
- Knowledge base retrieval failures or degraded performance
- Model endpoint unavailability or error rates
- Unusual query patterns indicating potential issues

Troubleshooting Framework

Common Issues and Solutions

Inaccurate Responses:

- **Symptom:** Assistant provides incorrect or outdated information
- **Investigation:** Verify knowledge base content accuracy and recency
- **Resolution:** Update source documents, adjust similarity thresholds, improve content chunking

Missing Source Citations:

- **Symptom:** Responses lack proper source attribution
- **Investigation:** Check retrieval configuration and citation generation settings
- **Resolution:** Verify knowledge base metadata, adjust retrieval parameters

Slow Response Performance:

- **Symptom:** Response latency exceeds acceptable thresholds
- **Investigation:** Monitor model inference time, retrieval latency, and resource utilization
- **Resolution:** Optimize retrieval parameters, scale model resources, implement response caching

Inappropriate Escalation Behavior:

- **Symptom:** Assistant escalates queries it should handle or fails to escalate complex issues
- **Investigation:** Review ruleset configuration and escalation trigger logic
- **Resolution:** Refine behavioral guidelines, adjust confidence thresholds

Diagnostic Procedures**Response Quality Assessment:**

1. Compare responses to documented correct answers
2. Validate source attribution accuracy
3. Check response coherence and professional tone
4. Verify compliance with organizational guidelines

Performance Analysis:

1. Measure end-to-end response latency
 2. Analyze component-specific performance contributions
 3. Monitor resource utilization patterns
 4. Evaluate scaling behavior under load
-

Production Deployment**Deployment Readiness Checklist****Technical Validation:**

- ☐ Comprehensive testing across all supported query types
- ☐ Performance validation under expected load conditions
- ☐ Integration testing with organizational systems
- ☐ Security review and compliance verification

Operational Preparation:

- ☐ Monitoring and alerting configuration
- ☐ Support procedures and escalation workflows

- ☐ Documentation for maintenance and troubleshooting
- ☐ User training and adoption planning

Continuous Improvement Framework

Performance Monitoring:

- Regular analysis of user interactions and satisfaction metrics
- Knowledge base effectiveness through retrieval analytics
- Model performance trends and optimization opportunities

Content Management:

- Systematic review and update of knowledge base content
- Gap analysis based on unresolved customer queries
- Integration of new documentation and policy updates

System Evolution:

- Model upgrade evaluation and testing procedures
 - Feature enhancement based on user feedback
 - Integration opportunities with additional organizational systems
-

Next Steps and Advanced Capabilities

Capability Expansion

Tool Integration: Connect external systems for account lookups, ticket creation, and workflow automation using *Tools Reference Manual*.

Multi-Modal Support: Extend capabilities to handle document uploads, image queries, and voice interactions.

Advanced Analytics: Implement conversation analytics for customer insights and service optimization.

Learning Resources

Foundational Concepts:

- *Learning Paths* for comprehensive AI Factory understanding
- *Pipelines reference - Preparers* for advanced content processing

Operational Excellence:

- Model serving optimization and scaling strategies
 - Advanced retrieval techniques for complex knowledge domains
 - Integration patterns with existing customer service infrastructure
-

This implementation guide provides a comprehensive foundation for deploying production-ready customer service assistants while maintaining organizational control over data, models, and operational procedures.

How to operate and observe

How-To View and Manage Threads in Gen AI Builder in Hybrid Manager

Prerequisite: Access to the Hybrid Manager UI with AI Factory enabled. See </edb-postgres-ai/1.3/hybrid-manager/ai-factory/>.

This guide provides systematic procedures for accessing and analyzing conversation threads within Gen AI Builder to support quality assurance, compliance monitoring, and assistant optimization activities.

Prerequisites

System Requirements

- Gen AI Builder access with appropriate thread viewing permissions
- At least one completed assistant conversation generating thread data
- Understanding of assistant configuration and expected behavior patterns

User Roles This procedure applies to:

- Quality assurance teams evaluating assistant performance
- Compliance officers conducting conversation audits
- Developers debugging assistant behavior issues
- AI architects analyzing interaction patterns

Access Thread Management

1. Navigate to Thread Interface

Access the **Threads** section within Gen AI Builder console. Thread management typically appears within the main navigation structure or under application management categories.

2. Review Thread Catalog

The thread catalog displays comprehensive conversation metadata:

Thread Identification

- **Name:** User-assigned or system-generated thread identifier
- **Alias:** Unique system identifier for programmatic access
- **Created:** Initial thread creation timestamp
- **Updated:** Most recent message or modification timestamp

Navigation Controls

- **Search Functionality:** Locate threads using keywords, aliases, or metadata filters
- **Pagination:** Navigate large thread collections efficiently
- **Sorting Options:** Order threads by creation date, activity, or other criteria

Thread Analysis Procedures

3. Locate Target Thread

Search Strategies

- **Keyword Search:** Use conversation content or topic keywords
- **Temporal Filtering:** Filter by creation or update date ranges
- **Assistant Filtering:** Narrow results to specific assistant instances
- **User Identification:** Search by user identifiers or session data

Search Examples

```
Search Terms:

- "account opening inquiry"

- Assistant: "customer_service_bot"

- Date Range: "Last 7 days"

- Thread Alias: "thread_abc123"
```

4. Access Thread Details

Select thread from catalog to open detailed conversation view containing:

Thread Overview Panel

- Complete thread metadata and identification information
- Thread lifecycle timestamps and modification history
- Associated assistant configuration and version information

Action Controls

- Thread management operations (edit, delete, export)
- Sharing and collaboration features
- Archival and retention controls

5. Analyze Conversation Flow

Message Sequence Analysis Review conversation progression through systematic examination:

1. **Message Navigation:** Step through individual messages using sequence controls
2. **Content Inspection:** Examine full message content and formatting
3. **Metadata Review:** Analyze message timestamps, processing duration, and system information
4. **Tool Execution Tracking:** Identify tool invocations and result integration

Message Detail Examination For each conversation turn, evaluate:

User Input Analysis

- Intent clarity and specificity

- Context preservation across conversation turns
- Information completeness and accuracy

Assistant Response Evaluation

- Response relevance and accuracy
- Source attribution and citation quality
- Tone consistency with configured rulesets
- Knowledge base utilization effectiveness

Tool Integration Assessment

- Tool selection appropriateness for user requests
- Parameter extraction accuracy from conversation context
- Result integration and presentation quality
- Error handling during tool execution failures

6. Performance Assessment

Quality Metrics Evaluation

Assessment Area	Evaluation Criteria
Intent Recognition	Accuracy of user request interpretation
Knowledge Retrieval	Relevance and accuracy of information sources
Response Quality	Coherence, completeness, and helpfulness
Behavioral Compliance	Adherence to configured rulesets and policies
Tool Utilization	Appropriate tool selection and execution

Behavioral Analysis

- **Ruleset Compliance:** Verify adherence to configured behavioral guidelines
- **Tone Consistency:** Evaluate communication style alignment with organizational standards
- **Escalation Handling:** Review appropriate escalation trigger recognition
- **Privacy Protection:** Confirm appropriate handling of sensitive information

Thread Management Operations

7. Execute Management Actions

Thread Modification

- **Name Updates:** Modify thread identifiers for improved organization
- **Metadata Enhancement:** Add classification tags or organizational labels
- **Access Control:** Adjust viewing permissions based on content sensitivity

Data Management

- **Export Operations:** Generate thread exports for external analysis or archival
- **Retention Management:** Apply organizational data retention policies

- **Deletion Procedures:** Remove threads following established data governance protocols

Quality Assurance Workflow

1. Document performance assessment findings
2. Identify improvement opportunities for assistant configuration
3. Report compliance violations or policy adherence issues
4. Recommend knowledge base updates or ruleset modifications

Practical Example: Financial Services Inquiry Analysis

Scenario Context Customer inquiry regarding new savings account opening processed by financial services assistant.

Analysis Procedure

Thread Location

1. Navigate to Threads catalog
2. Search parameters: `savings account opening`
3. Filter by assistant: `financial_services_bot`
4. Select relevant thread from results

Conversation Evaluation

1. **Initial Request Assessment**
 - User intent: Account opening information request
 - Context: New customer inquiry with specific product interest
2. **Assistant Response Analysis**
 - Product information accuracy verification
 - Compliance with financial services communication requirements
 - Knowledge base utilization for current product details
3. **Multi-turn Interaction Review**
 - Follow-up question handling effectiveness
 - Context preservation across conversation turns
 - Appropriate information gathering for account opening process

Quality Assessment Documentation

- **Strengths:** Accurate product information delivery, compliant communication tone
- **Areas for Improvement:** Enhanced eligibility requirement explanation
- **Recommendations:** Knowledge base update with current promotional offers

Troubleshooting Common Issues

Thread Access Problems

Thread Not Found

- Verify search parameters accuracy and scope
- Confirm thread creation completion (processing delays possible)
- Check user permissions for thread visibility

Incomplete Message Data

- Validate conversation completion status
- Verify tool execution logging configuration
- Check system processing status for recent conversations

Analysis Challenges

Performance Issues

- **Response Accuracy Problems:** Review knowledge base content coverage and retrieval configuration
- **Behavioral Violations:** Examine ruleset configuration and enforcement mechanisms
- **Tool Execution Failures:** Analyze tool configuration and external system connectivity

Resolution Strategies

- Document specific issues with conversation context
- Cross-reference assistant configuration for troubleshooting
- Escalate systematic problems to development teams
- Implement temporary workarounds while addressing root causes

Operational Best Practices

Systematic Analysis Approach

Regular Review Procedures

- Establish consistent thread review schedules
- Implement standardized evaluation criteria
- Document findings systematically for trend analysis
- Maintain assessment records for compliance purposes

Quality Improvement Integration

- Use thread analysis findings for assistant optimization
- Identify patterns requiring knowledge base updates
- Recommend ruleset modifications based on behavioral analysis
- Coordinate with development teams for systematic improvements

Data Management Considerations

Privacy and Security

- Implement appropriate access controls for sensitive conversations
- Follow organizational data retention policies
- Secure export procedures for audit and analysis purposes
- Maintain confidentiality during review processes

Compliance Requirements

- Document review procedures for regulatory compliance
- Maintain audit trails for thread access and analysis activities
- Implement systematic approaches for compliance violation reporting
- Coordinate with legal and compliance teams for policy adherence

Related Documentation

Thread Analysis Resources

- *Threads Reference Manual* : Comprehensive thread system overview
- *Assistants Reference Manual* : Assistant optimization procedures
- *Rulesets Manual* : Behavioral compliance assessment methods

System Integration

- Gen AI Builder Overview : Platform architecture and component relationships
- *Hybrid Manager integration* : Infrastructure management procedures

3.5 AI Factory Pipelines

AI Factory Pipelines is a core capability of **EDB Postgres® AI** (EDB PG AI), enabling intelligent, automated AI data workflows directly in your Postgres clusters — including **Retrieval-Augmented Generation (RAG)**, **Knowledge Bases**, and vector-powered applications.

Pipelines abstracts away complexity in preparing, managing, and serving AI data for Postgres — integrating with **Vector Engine** and enabling **Sovereign AI** patterns where your data stays inside your database or trusted object storage.

3.5.1 Why Pipelines?

Modern AI workloads — especially RAG and Gen AI — require:

- Clean, preprocessed data
- Consistent embeddings
- Efficient vector indexes
- Real-time updates when source data changes

Manually managing these steps is complex and error-prone. Pipelines automates this lifecycle:

- Prepares and cleans data
- Runs embedding generation

- Maintains vector indexes
- Supports auto-processing and Knowledge Bases

Pipelines helps eliminate “hallucinations” from stale embeddings and makes building reliable Gen AI easier.

3.5.2 What You Can Do

With Pipelines, you can:

- Automate **vector embedding pipelines** for text and image data
- Maintain **Knowledge Bases** with real-time or batch updates
- Perform **semantic and similarity search** using *Vector Engine (PGvector)*
- Integrate with **Gen AI Builder** and other AI Factory components
- Build fully auditable, controlled **Sovereign AI** systems on your data

3.5.3 How It Works

Preparers

Preparers define how source data is cleaned, chunked, and prepared for embedding. You can create preparers for:

- Postgres tables (`aidb.create_table_preparer()`)
- Object storage (S3-compatible) (`aidb.create_volume_preparer()`)

Once created, you can run:

- **Bulk processing** (`aidb.bulk_data_preparation()`)
- **Auto-processing** (`aidb.set_auto_preparer()`) to keep embeddings up to date automatically.

See *Pipelines reference - Preparers* to learn more.

Knowledge Bases

Knowledge Bases provide a simple abstraction for RAG:

- Define a Knowledge Base (`aidb.create_table_knowledge_base()` , `aidb.create_volume_knowledge_base()`)
- Automatically manage embeddings and vector indexes
- Use `aidb.retrieve_key()` and `aidb.retrieve_text()` for semantic search

Supports:

- Text and image data
- Multi-modal models (e.g., CLIP)
- Integration with Gen AI Builder’s retrieval flows

See Knowledge Base Concepts .

Auto-Processing

Pipelines supports flexible auto-processing modes:

- Sync or async
- Table or volume sources

- Configurable batch sizes and triggers

See *Pipeline auto-processing* for details.

3.5.4 Who Should Use Pipelines?

Pipelines is ideal for:

- Gen AI Builders creating **enterprise Knowledge Bases**
- Data Engineers building **semantic search pipelines**
- AI teams needing **automated RAG pipelines**
- Architects implementing **Sovereign AI** in regulated environments

3.5.5 When to Use Pipelines

Use Pipelines when you need to:

- Keep embeddings current with changing data
- Reduce RAG errors caused by stale or incomplete embeddings
- Power multi-modal Gen AI search across text and images
- Maintain full auditability and governance of embedding processes

3.5.6 Configuration values explained

- **Chunk size and overlap:** Smaller chunks improve grounding; overlap helps preserve context across chunks. Tune based on document structure.
- **Embedding model selection:** Use a consistent model for ingestion and query. Choose for your domain and language.
- **Auto-processing cadence:** Align with data change frequency and SLA. Consider sync vs. async modes.
- **Batch sizes and parallelism:** Increase for throughput but monitor storage, network, and memory.
- **Similarity threshold and top-K:** Higher thresholds reduce noise; tune top-K for recall vs. latency.

3.5.7 Learn More

- Knowledge Base Concepts
- *Pipelines reference - Preparers*
- *Pipeline auto-processing*
- *Supported Embedding Models*
- *Vector Engine (PGvector)*

AI Accelerator Pipelines helps you build trusted, scalable **Sovereign AI** solutions on top of your Postgres data — keeping AI pipelines clean, current, and production-ready.

Start here

AI Accelerator Pipelines overview

As part of EDB Postgres AI - AI Accelerator, Pipelines abstracts away the complexity of working with AI data. It transforms Postgres into a powerful platform for AI data management, combining vector search from PGvector with automation for complex AI workflows.

Pipelines, auto-processing, and intelligent knowledge base

Pipelines handles the entire lifecycle of data preparation through embedding generation, storage, and indexing. You can select your desired model from a list of supported models or connect to any OpenAI API-compatible external model. You can then let Pipelines manage everything else from data ingestion to efficient similarity search.

With data preparation, Pipelines can perform common preprocessing steps within the AI Accelerator directly from Postgres. Pipelines can even perform this pre-processing preparation automatically, keeping processed data up to date.

With auto-processing, Pipelines automates creating and updating embedding, ensuring that vector stores remain up to date without manual embedding management. This approach reduces the risk of stale data that causes GenAI errors and hallucinations.

The intelligent knowledge base feature lets you perform similarity and semantic searches across text and image data with a single knowledge base function call. These searches are possible whether the data is stored in Postgres or object storage.

How Pipelines works

Pipelines delivers its functionality through the Pipelines aidb extension that's embedded into the Postgres server. The extension provides a set of functions and views that allow you to interact with the AI data in your Postgres database.

Preparers

The Pipelines aidb extension introduces the concept of a *preparer* that you can create for a given type and location of source AI data. This data can reside either in regular columns of a Postgres table or in an S3-compatible object storage bucket.

A preparer encapsulates the functionality to perform common preprocessing steps on source data for use in embeddings generation. The application just needs to create a preparer via the `aidb.create_table_preparer()` function for Postgres table sources or `aidb.create_volume_preparer()` for source data stored externally on S3 or on local filesystems.

By default, there is no automatic processing so the application can trigger bulk data preparation processing for all existing data in the source location using `aidb.bulk_data_preparation()`. Automatic processing can also be configured using `aidb.set_auto_preparer()` for source data in Postgres tables. The output from the preparer(s) can then be fed into the input for knowledge bases.

Knowledge bases

The Pipelines aidb extension introduces the concept of a *knowledge base* that you can create for a given type and location of AI data. This data can reside either in regular columns of a Postgres table or in an S3-compatible object storage bucket. Currently, Pipelines supports unstructured plain text documents as well as a set of image formats.

A knowledge base encapsulates all processing that's needed to make the AI data in the provided source location searchable and retrievable through similarity. To create a knowledge base, the application uses the `aidb.create_table_knowledge_base()` function for Postgres tables or `aidb.create_volume_knowledge_base` for stored data externally on S3 or local file systems.

Auto-processing

Auto-processing has broad configurability including synchronous and asynchronous as well as batch processing. Availability of these features depends on the type of pipeline (knowledge base or preparer) and the data source and destination type (table or volume).

See [Pipeline auto-processing](#) for more details.

Intelligent knowledge base

Once a knowledge base is created and all embeddings are up to date, the application can use `aidb.retrieve_key()` to run a similarity search and retrieval by providing a query input. It returns the id of the matched items. When the knowledge base is created for text data, the query input is also a text term. For image knowledge bases, the query input is an image. If you're using a multi-modal model with your knowledge base, such as CLIP, you can use text or images as source and query inputs. This approach allows you to search for images using text and vice versa.

The `aidb` knowledge base makes sure to use the same encoder LLM for the query input as used for the embedding, conducts a similarity search, and returns the ranked list of similar data from the source location. For Postgres tables, you can also use `aidb.retrieve_text()` to retrieve the actual data from the source table.

Pipelines currently supports a broad list of open encoder LLMs from HuggingFace as well as a set of OpenAI encoders. HuggingFace LLMs run locally on the Postgres node, while OpenAI encoders involve a call out to the OpenAI cloud service. The `aidb.model_providers` table shows the list of currently supported model providers.

Compatibility

aidb

Supported platforms

- Ubuntu 22.04LTS and 24.04LTS on X86/64.
- Debian 12 (Bookworm) on X86/64 and ARM64.
- Redhat/RHEL 9/8 on X86/64.
- Redhat/RHEL 9 on ARM64.

Not currently supported

- SLES.
- Debian before the current version 12.
- Non-Linux platforms.

Supported PostgreSQL versions

- EDB Postgres Advanced Server Version 14, 15, 16, and 17.
- EDB Postgres Extended Version 14, 15, 16, and 17.
- PostgreSQL 14, 15, 16, and 17.

pgfs

Supported platforms

- Ubuntu 22.04LTS and 24.04LTS on X86/64.
- Debian 12 (Bookworm) on X86/64 and ARM64.
- Redhat/RHEL 9/8 on X86/64.
- Redhat/RHEL 9 on ARM64.

Not currently supported

- SLES.
- Debian before the current version 12.
- Non-Linux platforms.

Supported PostgreSQL versions

- EDB Postgres Advanced Server Version 14, 15, 16, and 17.
- EDB Postgres Extended Version 14, 15, 16, and 17.
- PostgreSQL 14, 15, 16, and 17.

Limitations

Blocking operations

Preparer pipelines currently don't support asynchronous operations. All operations are blocking and take place in the foreground. When executing bulk data preparation, the system will be blocked until the embedding is complete for all the items currently in the database or storage.

Observability

Preparer pipelines currently have limited observability. Refer to the Postgres server logs and output of the commands. There isn't a single pane of glass for monitoring the entire system.

Large documents

Pipelines currently doesn't handle **chunking large documents**.

Data filtering

While Pipelines can limit embedded documents using SQL filters and views based on the content of rows, it currently doesn't support filtering on data in S3 storage. It's limited to using subpaths and prefix filtering.

Load balancing for models

There's currently no load balancing mechanism for model access.

Data formats

Knowledge base pipelines currently supports only text and image formats. Other formats, including structured data, video, and audio, aren't currently supported.

Auto-processing

- Live auto-processing is supported only for text data. Image data embeddings can be manually computed or using background auto-processing.
- Background auto-processing is supported only by the knowledge base pipeline, not by the preparer pipeline.

Upgrading

When upgrading the aidb and pgfs extensions, there's currently no support for Postgres extension upgrades. When upgrading to a new version of the extensions, you must therefore drop and re-create the extensions.

Refer to *Upgrading AI Accelerator Pipelines* for more details and commands.

Get started

Getting started with Pipelines

Where to start

The best place to start is with the *AI Accelerator Pipelines overview* to get an understanding of what Pipelines is and how it works.

Installation

Pipelines is included with the EDB Postgres AI - AI Accelerator suite of tools. To install Pipelines, follow the instructions in the AI Accelerator Installation Guide .

Using Pipelines

After you have Pipelines installed, you can start using it to work with your data.

Log into your Postgres server and ensure the Pipelines extension is installed:

```
CREATE EXTENSION aidb CASCADE;
```

This example works solely with Postgres table data, so you don't need to install the pgfs extension.

For the example, create a table to work with:

```
CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  product_name TEXT NOT NULL,
  description TEXT,
  last_updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
__OUTPUT__
CREATE TABLE
```

Then insert some data into it:

```
INSERT INTO products (product_name, description) VALUES
  (Hamburger, A delicious combination of bread and meat),
  (Cheesburger, Improving on a classic, the cheese brings favorite flavors),
  (Fish n Chips, The fish is a little greasy and the chips do not help),
  (Fries, Never sure about these on their own, needs seasoning),
  (Burrito, Always ready for this parcel of edible wonder),
  (Pizza, It is very much a staple, but the rolled dough with toppings does not
↳inspire),
  (Sandwich, The blandest of offerings, the sandwich is predominantly boring
↳bread),
  (Veggie Burger, The ultra-processed vegetable product in this is neither
↳healthy nor delicious),
```

(continues on next page)

(continued from previous page)

```

      (Kebab, Maybe one of the great edible treats, sliced lamb, salad and crisp_
→pitta);
__OUTPUT__
INSERT 0 9

```

The result is a table with some data in it: food products and some very personal opinions about them.

!!!Note

This example uses simple, processed data. Pipelines can also handle multiple preprocessing steps via [Pipelines reference - Preparers](#).

The core of Pipelines is the *knowledge base*. A knowledge base is a way to access the data in the table and use it in AI workflows.

Getting started with Pipelines

See also

In [Pipelines models reference](#), you can learn how to create more models with Pipelines, including external models from OpenAI API-compatible services.

In [Pipelines reference - Preparers](#), you can learn more about how to create preparers to perform preprocessing steps on your data before embeddings are calculated.

In [Knowledge Bases](#), you can learn more about how to use knowledge bases with external data sources, local files, or S3 storage and how to use the knowledge base functions to get the data you need.

Installing AI Accelerator Pipelines

Pipelines is delivered as a set of extensions. Depending on how you're deploying Pipelines, these extensions can be installed by your deployment platform provider or already installed. Contact your deployment platform provider for details. If you deploy your own Postgres server, you need to install the extensions manually.

- [AI Accelerator Pipelines](#)

Once the packages are installed, you can [Completing and verifying the extension installation](#) by activating the extensions in Postgres.

If you have an existing installation and want to upgrade to a new version, see [Upgrading AI Accelerator Pipelines](#).

AI Accelerator Pipelines

Installing the extension packages

To use AI Accelerator Pipelines, you need three extension packages:

- `aidb` — AI Database extension.
- `pgfs` — Postgres File System (PGFS) extension.
- `vector` — PGvector extension.

Prerequisite

Before you start installing the packages, you need to have a PostgreSQL server running. You can install AI Accelerator Pipelines on a community PostgreSQL server, an EDB Postgres Advanced Server server, or an EDB Postgres Extended server. If you need help setting up a server, consult the installation instructions for those products. When you install the server, make a note of the package name used as you will need it to install the AI Accelerator Pipelines packages.

- For community PostgreSQL, the package name is `postgresql-<version>`.
- For EDB Postgres Advanced Server, the package name is `edb-as<version>` followed by `-server`.
- For EDB Postgres Extended, the package name is `edb-postgresextended-<version>` followed by `-server`.

Once you have established the package name, you can install the server using the package manager for your operating system.

Installing the packages

Once the server is running, you need to install several packages. The packages are available in the EDB repository and can be installed using the appropriate package manager for your operating system. Use:

- `apt` for Debian/Ubuntu-based systems with `.deb` packages
- `dnf` for Red Hat-based systems with `.rpm` packages

The package name depends on your operating system.

For community PostgreSQL

The packages are:

- `edb-pg<version>-aidb` — AI Database extension.
- `edb-pg<version>-pgfs` — Postgres File System extension.

For the PGvector extension, the packages are:

- rpm-based systems: `edb-pg<version>-pgvector0`
- deb-based systems: `edb-pg<version>-pgvector-0`

For EDB Postgres Advanced Server

The packages are:

- `edb-as<version>-aidb` — AI Database extension.
- `edb-as<version>-pgfs` — Postgres File System extension.

For the PGvector extension, the package depends on your package format:

- For rpm-based systems: `edb-as<version>-pgvector0`
- For deb-based systems: `edb-as<version>-pgvector-0`

For EDB Postgres Extended

The packages are:

- `edb-pgextended-<version>-aidb` — AI Database extension.
- `edb-pgextended-<version>-pgfs` — Postgres File System extension.

For the PGvector extension, the package depends on your package format:

- For rpm-based systems: `edb-postgresextended<version>-pgvector0`
- For deb-based systems: `edb-postgresextended<version>-pgvector-0`

Example

The command for installing the packages on Ubuntu 22.04 with EDB Postgres Advanced Server 18 looks like this:

```
sudo apt-get install edb-as17-aidb edb-as17-pgfs edb-as17-pgvector-0
```

With the packages installed, you can connect to the database and [Completing and verifying the extension installation](#).

Completing and verifying the extension installation

Installing the AI Database extension

The AI Database extension provides a set of functions to run AI models in the database. Install the extension using the `CREATE EXTENSION` command:

```
edb=# CREATE EXTENSION aidb CASCADE;
__OUTPUT__
NOTICE: installing required extension "vector"
CREATE EXTENSION
```

Additional steps for EDB Postgres Distributed (PGD)

A manual step is required to set up replication within the PGD cluster for the AIDB extension catalog tables:

```
bdrdb=# SELECT aidb.bdr_setup();
__OUTPUT__
 bdr_setup
-----
(1 row)
```

You can confirm that the AIDB tables are part of the desired replication set using this SQL command:

```
bdrdb=# select * from bdr.tables where nspname=aidb;
__OUTPUT__
 relid | nspname |          relname          | set_name |
-----+-----+-----+-----+
  set_ops |          | rel_columns | row_filter | conflict_detection
-----+-----+-----+-----+
 18281 | aidb    | pipeline_runtime_state    | p-2y7357q195-a | {INSERT,
  UPDATE,DELETE,TRUNCATE} | row_origin
 18354 | aidb    | preparer_registry         | p-2y7357q195-a | {INSERT,
  UPDATE,DELETE,TRUNCATE} | row_origin
 18363 | aidb    | preparer_registry_source_table | p-2y7357q195-a | {INSERT,
  UPDATE,DELETE,TRUNCATE} | row_origin
 18375 | aidb    | preparer_registry_source_volume | p-2y7357q195-a | {INSERT,
  UPDATE,DELETE,TRUNCATE} | row_origin
 18430 | aidb    | knowledge_base_registry   | p-2y7357q195-a | {INSERT,
  UPDATE,DELETE,TRUNCATE} | row_origin
 18440 | aidb    | knowledge_base_registry_source_table | p-2y7357q195-a | {INSERT,
  UPDATE,DELETE,TRUNCATE} | row_origin
 18452 | aidb    | knowledge_base_registry_source_volume | p-2y7357q195-a | {INSERT,
  UPDATE,DELETE,TRUNCATE} | row_origin
(7 rows)
```

Setting up background workers

To use background processing with AIDB pipelines, the extension must be registered as a “preload library” in the Postgres config file.

Add the following setting to the `postgresql.conf` file. The location of the config file depends on your environment. You can identify the location with `SHOW config_file;`.

```
shared_preload_libraries = aidb
```

Proxy settings

If you’re using an HTTP proxy, you may need to set the `HTTP_PROXY` and `HTTPS_PROXY` environment variables in the Postgres environment. You can do this by adding the lines shown in the example that follows to the `environment` file in the `main` directory of the Postgres data directory.

For example, on Ubuntu with community Postgres, the `environment` file is located at `/etc/postgresql/16/main/environment`. Run the following commands to add the proxy settings to the `environment` file, and restart the Postgres service:

```
echo "HTTP_PROXY = http://<yourproxysettings>/" | sudo tee -a /etc/postgresql/16/main/
environment
echo "HTTPS_PROXY = http://<yourproxysettings>/" | sudo tee -a /etc/postgresql/16/
main/environment
sudo systemctl restart postgresql@16-main
```

Replace `<yourproxysettings>` with your proxy settings. If you’re using a different version of Postgres, replace `16` with your version number. Consult the documentation for your Postgres distribution for the location of the `environment` file.

Airgapped environments aren’t currently supported.

Installing the Postgres File System extension

The Postgres File System (PGFS) extension provides a set of functions to interact with the file system from the database. Install the extension using the `CREATE EXTENSION` command:

```
edb=# create extension pgfs;
CREATE EXTENSION
```

Validating the installation

You can check whether the extensions were installed by running the `\dx` command in `psql`:

```
edb=# \dx
__OUTPUT__

                                List of installed extensions
  Name      | Version | Schema | Description
-----+-----+-----+-----
aidb        | 2.1.1   | aidb   | aidb: makes it easy to build AI
applications with postgres
pgfs        | 1.0.6   | pgfs   | pgfs: enables access to filesystem-like
storage locations
```

(continues on next page)

(continued from previous page)

```
vector          | 0.8.0   | public      | vector data type and ivfflat and hnsw_
↪access methods
```

In addition to the `aidb`, `pgfs`, and `vector` extensions listed in the output, other extensions are typically listed as well.

Upgrading AI Accelerator Pipelines

From version 4.3.0 onwards, AI Accelerator supports in-place updates for the extension. Once the new version is installed you can run the `ALTER EXTENSION` command to update the extension in your Postgres database.

```
ALTER EXTENSION aidb UPDATE;
```

If you are upgrading from a version prior to 4.3.0, you need to follow the steps below to ensure a clean upgrade. This is because the extension was not designed for in-place updates before this version.

Upgrading the Pipelines Extension (aidb)

!!!Note This doesn't delete any managed data. It just ensures that no dangling references (e.g. Triggers) remain while AIDB is uninstalled, and allows you to do a clean setup after upgrading.

1. Delete all configured Pipelines

Get a list of all the knowledge base and preparer pipelines you have configured: `SELECT name FROM aidb.knowledge_bases;` and `SELECT name FROM aidb.preparers;` . Then delete all knowledge bases and preparer by calling `SELECT aidb.delete_knowledge_base('name');` and `SELECT aidb.delete_preparer('knowledge_base__5887');` .

You can use this SQL code to do both in one operation:

```
WITH names AS (SELECT name FROM aidb.knowledge_bases)
  SELECT aidb.delete_knowledge_base(name) FROM names;

WITH names AS (SELECT name FROM aidb.preparers)
  SELECT aidb.delete_preparer(name) FROM names;
```

2. Uninstall the old version of the extension

```
edb=# DROP EXTENSION aidb CASCADE;
DROP EXTENSION
edb=#
```

3. Install the new version of the extension in your environment

Follow the same steps you took to install the old version: *AI Accelerator Pipelines* .

4. Create the extension in your Postgres Database

```
edb=# CREATE EXTENSION aidb CASCADE;
NOTICE: installing required extension "vector"
CREATE EXTENSION
edb=#
```

5. Re-create the Pipelines you want to use

Run the same commands you originally ran to create the Pipelines.

Upgrading PGFS

If you are using PGFS, you will need to drop and re-create the extension after upgrading.

!!!Note This will delete references to your PGFS storage locations; it will not delete the actual data. Storage locations need to be re-created after upgrading.

```
DROP EXTENSION pgfs CASCADE;
CREATE EXTENSION pgfs CASCADE;
```

AI Accelerator Permissions Setup

A new database role, `aidb_users`, is created as part of the AIDB extension installation. Membership to this role is required for access to most extension functions. This role will not be removed if the extension is deleted.

`aidb_users` is designed to simplify access management for AIDB features:

- **Purpose:** Granting a user the `aidb_users` role provides access to all AIDB functions, allowing access to the extension's capabilities.
- **Security:** The `aidb_users` role does **not** grant access to AIDB's internal tables, ensuring that only the extension's public API is exposed to users.
- **Background Workers:** When background workers are used, they execute as the user who defined them, or as a role specified by the user—provided the user has access to that role.

To allow a user to use AIDB functions, simply grant them the `aidb_users` role:

```
GRANT aidb_users TO your_user;
```

Here is an example of creating a user named `alice` with no privileges or role membership to start with:

```
CREATE ROLE alice
  LOGIN
  PASSWORD change_me;  -- replace with a real password
```

Now add `alice` to the application group role `aidb_users`

```
GRANT aidb_users TO alice;
```

`alice` now inherits whatever global privileges `aidb_users` has (for example: EXECUTE on routines in `aidb`, USAGE on schemas, etc.)

Give `alice` (not `aidb_users`) full rights on selected tables in public schema. This example shows three arbitrary tables. These would ordinarily be source tables used in the pipeline

```
GRANT SELECT, INSERT, UPDATE, DELETE
  ON public.documents,
     public.email_text,
     public.chats
  TO alice;
```

And in cases where you need `alice` to be able to create objects in `public`:

```
GRANT CREATE ON SCHEMA public TO alice;
```

Alice automatically owns whatever she creates, so she can ALTER or DROP her own objects. She cannot drop or alter other users' objects in `public`.

Now you may Verify privileges ``sql **:raw-latex:`z`public.documents, :raw-latex:`z`public.email_text, :raw-latex:`z`public.chats**

Core concepts

AI Accelerator Pipelines knowledge bases

Knowledge bases can be used for semantic search or for retrieval augmented generation (RAG).

A configured knowledge base can compute embeddings (vectors) for your input data. Once this is done, the knowledge base offers a number of interfaces to return data based on a query.

Knowledge bases concepts

Knowledge bases can be used for semantic search or for retrieval augmented generation (RAG). A configured knowledge base can compute embeddings (vectors) for your input data. After the computing is done, the knowledge base offers interfaces to return data based on a query.

Data sources and data types

A data source is where the data to be embedded/retrieved comes from. The aidb extension supports two types of data sources:

- **Table** — A column in a table in the PG database.
- **Volume** — An AIDB *volume*, which is a wrapper for accessing an S3 object store or local file system through the PGFS extension.

Data types are:

- Text
- Image

You can combine any data source with any data type:

- **Table and text:** The text, in this case, is just a column with TEXT/VARCHAR type in a Postgres table. An example is the description for a list of products.
- **Table and image:** The image, in this case, would be stored in a BYTEA column in the table, which would hold the bytes of the image. Pipelines doesn't currently support large objects (also known as BLOB) types.
- **Volume and text:** An example is text files in an S3 bucket.
- **Volume and image:** The images can be stored as objects in an S3 bucket.

Embedding

Knowledge bases always need a *vector table* to store the vector embeddings that belong to the source data. Pipelines uses a model with *encoder* capabilities to generate the vectors for the source data.

Each model has a different *dimensionality*, that is, the number of dimensions of the vector. (The vector [1, 2, 2] has 3 dimensions.) Pipelines uses PGvector *vector* column types to store the embeddings. Those have a fixed number of dimensions. For this reason, the encoder interface on the models returns the number of dimensions the vectors will have.

Pipelines creates the vector table and also supports using an existing one. The vector column dimensionality must fit the model.

Consistency with source data

To retrieve correct and consistent data, embeddings must be in sync with the source data. Pipelines supports different *Pipeline auto-processing* modes achieve this:

- Live auto-processing sets up Postgres Triggers on the source data to guarantee consistency
- Background auto-processing asynchronously processes changes in the background
- Disabled auto-processing allows you to manually run processing as needed with a single function call.

Pipelines knowledge bases usage

Creating a new knowledge base configuration

There are two types of knowledge bases: table and volume. Given the different nature of the data sources and the options required for each, there are different functions to create them.

Knowledge base for a table data source

The `aidb.create_table_knowledge_base()` function is used to create a knowledge base for a table data source. Below you can find a minimal example. See [aidb.create_table_knowledge_base](#) for a full list of options.

Example: Creating a knowledge base

```
SELECT aidb.create_table_knowledge_base(
    name => test_knowledge_base,
    model_name => bert,
    source_table => test_source_table,
    source_data_column => content,
    source_data_format => Text
);
```

If you're using only Postgres tables, skip to [Creating the embeddings](#).

If you're using external data sources, you need to create a volume and create a knowledge base for it, which is explained next.

Knowledge base for a volume data source

Creating a new volume

Before you can create a knowledge base for a volume, you need to create a volume and a PGFS storage location. See [AIDB volumes for accessing PGFS storage locations](#) for more detail.

Here, we create a *Pipelines PGFS reference* for a public S3 bucket:

```
SELECT pgfs.create_storage_location(aidb_text_bucket, s3://aidb-rag-app,
    options => {"region": "eu-central-1", "skip_signature": "true"}
);
```

The `aidb.create_volume_knowledge_base` function is used to create a volume.

Example: Creating a volume

```
SELECT aidb.create_volume(aidb_text, aidb_text_bucket, /, Text);
```

The `server_name` comes from calling PGFS functions to create a storage location [pgfs.create_storage_location](#). The path is the path to the data in the storage location. The last argument indicates the data format; Text or Image.

Creating a knowledge base for a volume

The `aidb.create_volume_knowledge_base()` function is used to create a knowledge base for a volume data source. Below you can find a minimal example. See [aidb.create_volume_knowledge_base](#) for a full list of options.

Example: Creating a knowledge base for a volume

```
SELECT aidb.create_volume_knowledge_base(
    name => test_knowledge_base_volume,
    model_name => bert,
    source_volume_name => aidb_text
);
```

Creating the embeddings

You can use bulk embedding to manually run embedding computation:

```
SELECT aidb.bulk_embedding(test_knowledge_base);
```

and for the volume example:

```
SELECT aidb.bulk_embedding(test_knowledge_base_volume);
```

Enable live auto-processing (automated embeddings) for any future changes:

```
SELECT aidb.set_auto_knowledge_base(test_knowledge_base, Live);
```

You can disable auto-processing as well (It is 'Disabled' by default when creating new knowledge bases):

```
SELECT aidb.set_auto_knowledge_base(test_knowledge_base, Disabled);
```

Retrieving

A basic key knowledge base, `aidb.retrieve_key` is available. It doesn't look up the source data but returns the ID/key of the matching embeddings.

Retrieving the key

```
aidb.retrieve_key(<knowledge base name>, <query string>, <optional number of results>
→);
```

Example: Retrieving the key

```
SELECT * FROM aidb.retrieve_key(test_knowledge_base, shoes, 2);
```

__OUTPUT__

key	distance
43941	0.2938963414490189
19337	0.3023805122617119

(2 rows)

You can use this command if you want to do a join/lookup yourself based on the key. For knowledge bases with external (volume) data sources, this capability is especially useful. Usually the application itself wants to do the retrieval from the external data source. Or you might want to push down the actual retrieval to a client application.

Retrieving the text

The `retrieve_text` function joins the embeddings with the source data and directly returns the results:

```
aidb.retrieve_text(<knowledge base name>, <query string>, <optional number of results>
→);
```

Example

```
SELECT * FROM aidb.retrieve_text(test_knowledge_base, jacket, 2);
```

__OUTPUT__

key	value	distance
19337	United Colors of Benetton Men Stripes Black Jacket	0.2994317672742334
55018	Lakme 3 in 1 Orchid Aqua Shine Lip Color	0.3804609668507203

(2 rows)

Listing the knowledge bases

A view is available that lists all the knowledge_bases. `aidb.knowledge_bases` also includes some of the knowledge bases configuration:

```
SELECT * FROM aidb.knowledge_bases;
```

__OUTPUT__

id	name	vector_table_name	vector_table_key_	column	vector_table_vector_column	model_name	topk	distance_operator	options	source_table_name	source_table_data_column	source_table_data_column_	type	source_table_key_column	source_volume_name
2	test_knowledge_base	test_knowledge_base_vector	id	embeddings	simple_model	5	InnerProduct		{}	test_source_table	content	Text			
5	test_knowledge_base_cosa	test_knowledge_base_cosa_vector	id	embeddings	simple_model	1	L2		{}						

(continues on next page)

(continued from previous page)

```

→ | test_source_table | content | Text |
→ id |
3 | test_knowledge_base_cos | test_knowledge_base_cos_vector | id |
→ | embeddings | simple_model | 5 | Cosine | {} |
→ | test_source_table | content | Text |
→ id |
(3 rows)

```

We recommend that you select only the columns you're interested in:

```

SELECT name, source_table_name FROM aidb.knowledge_bases;
__OUTPUT__
      name | source_table_name
-----+-----
test_knowledge_base | test_source_table
test_knowledge_base_cos | test_source_table
test_knowledge_base_cosa | test_source_table
(3 rows)

```

Deleting a knowledge base

This command doesn't delete the vector table or anything else. It only deletes the configuration:

```
SELECT aidb.delete_knowledge_base(<name>);
```

End-to-end example

This *Knowledge bases end-to-end example usage* is an end-to-end example for a table/text knowledge base.

Knowledge bases end-to-end example usage

This is a full end-to-end example of using knowledge bases in EDB Postgres AI - AI Accelerator.

```

-- Create extension
DROP EXTENSION aidb CASCADE;
CREATE EXTENSION aidb CASCADE;

drop table if exists test_source_table_ajz72eb cascade;
drop table if exists test_knowledge_base_ajz72eb_vector cascade;

-- Create source test table
CREATE TABLE test_source_table_ajz72eb
(
    id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    content TEXT NOT NULL,
    unrelated_column TEXT
);
INSERT INTO test_source_table_ajz72eb
VALUES (43941, Catwalk Women Brown Heels),
(55018, Lakme 3 in 1 Orchid Aqua Shine Lip Color),
(19337, United Colors of Benetton Men Stripes Black Jacket);

```

(continues on next page)

(continued from previous page)

```

SELECT aldb.create_table_knowledge_base(
  name => test_knowledge_base_ajz72eb,
  model_name => bert, -- this is a pre-defined locally running model
  source_table => test_source_table_ajz72eb,
  source_data_column => content,
  source_data_format => Text
);

SELECT * FROM aldb.kbs; -- This is an alias for the view aldb.knowledge_bases. It
↳ shows the configuration for all knowledge bases

SELECT aldb.bulk_embedding(test_knowledge_base_ajz72eb); -- Since no auto-processing
↳ was configured, we manually run it

SELECT * FROM aldb.kbstat; -- This is an alias for aldb.knowledge_base_stats. It
↳ shows statistics about the number of records and vectors

-- Perform retrieval similarity search for the closest `key`
SELECT * FROM aldb.retrieve_key(test_knowledge_base_ajz72eb, orchid);
SELECT * FROM aldb.retrieve_key(test_knowledge_base_ajz72eb, orchid, 2); -- Limit to
↳ top 2 results

SELECT * FROM aldb.retrieve_text(test_knowledge_base_ajz72eb, orchid);
SELECT * FROM aldb.retrieve_text(test_knowledge_base_ajz72eb, orchid, 2); -- Limit
↳ to top 2 results

-- enable live auto-processing
SELECT aldb.set_auto_knowledge_base(test_knowledge_base_ajz72eb, Live);

-- add additional data to test auto-processing
INSERT INTO test_source_table_ajz72eb
VALUES (11211, Bicycle),
      (11311, What is this?),
      (11411, Elephants);

-- check embeddings
SELECT id FROM test_knowledge_base_ajz72eb_vector;

-- delete one of the source rows
DELETE FROM test_source_table_ajz72eb WHERE id = 11211;

-- check embeddings
SELECT id FROM test_knowledge_base_ajz72eb_vector;

```

Pipelines knowledge base performance tuning

Background

You can optimize performance (that is, throughput of embeddings per second) by changing pipeline and model settings.

Knowledge base pipelines process collections of individual records (rows in a table or objects in a volume). Rather than processing each record individually and sequentially, or processing all of them concurrently, AIDB offers batch

processing. All the batches get processed sequentially, one after the other. Within each batch, records get processed concurrently wherever possible.

- *Pipeline auto-processing* determines the number of records in each batch.
- Some model providers have configurable internal batch/parallel processing. We recommend leaving these setting at the default values and using the pipeline batch size to control execution.

Note

Vector indexing also has an impact on pipeline performance. You can disable the vector by using `index_type => 'disabled'` to exclude it from your measurements.

Testing and tuning performance

This example first sets up test data and a knowledge base pipeline, then measures and tunes the batch size.

1) Create a table and insert test data

The actual data content length has some impact on model performance. You can use longer text to test that.

```
CREATE TABLE test_data_10k (id INT PRIMARY KEY, msg TEXT NOT NULL);

INSERT INTO test_data_10k (id, msg) SELECT generate_series(1, 10000) AS id, hello_
↪world;
```

2) Create a knowledge base pipeline

The optimal batch size may be very different for different models. Measure and tune the batch size for each different model you want to use.

```
SELECT aidb.create_table_knowledge_base(
  name => perf_test,
  model_name => dummy,  -- use the model you want to optimize for
  source_table => test_data_10k,
  source_data_column => msg,
  source_data_format => Text,
  index_type => disabled, -- optionally disable vector indexing to include/exclude_
↪it from the measurement
  auto_processing => Disabled, -- we want to manually run the pipeline to measure_
↪the runtime
  batch_size => 100  -- this is the paramter we will tune during this test
);
__OUTPUT__
INFO:  using vector table: public.perf_test_vector
NOTICE:  index "vdx_perf_test_vector" does not exist, skipping
NOTICE:  auto-processing is set to "Disabled". Manually run "SELECT aidb.bulk_
↪embedding(perf_test);" to compute embeddings.
create_table_knowledge_base
- -----
perf_test
(1 row)
```

3) Run the pipeline, measure the performance

This test uses psql. (The `\timing on` command is a feature in psql.) If you use a different interface, find out how it can display timing information.

```
\timing on
__OUTPUT__
Timing is on.
```

Now run the pipeline:

```
SELECT aidb.bulk_embedding(perf_test);
__OUTPUT__
INFO: perf_test: (re)setting state table to process all data...
INFO: perf_test: Starting... Batch size 100, unprocessed rows: 10000, count(source_
↪records): 10000, count(embeddings): 0
INFO: perf_test: Batch iteration finished, unprocessed rows: 9900, count(source_
↪records): 10000, count(embeddings): 100
INFO: perf_test: Batch iteration finished, unprocessed rows: 9800, count(source_
↪records): 10000, count(embeddings): 200
...
INFO: perf_test: Batch iteration finished, unprocessed rows: 0, count(source_
↪records): 10000, count(embeddings): 10000
INFO: perf_test: finished, unprocessed rows: 0, count(source records): 10000,
↪count(embeddings): 10000
bulk_embedding
-----
(1 row)

Time: 207161,174 ms (03:27,161)
```

4) Tune the batch size

You can use this call to adjust the batch size of the pipeline. This example increases the batch size by 10x to 1000 records:

```
SELECT aidb.set_auto_knowledge_base(perf_test, Disabled, batch_size=>1000);
```

Run the pipeline again.

Note

When using a Postgres table as the source, with auto-processing disabled, AIDB has no means to detect changes in the source data. So each `bulk_embedding` call has to reprocess everything.

This is convenient for performance testing.

If you want to measure performance with a volumes source, delete and re-create the knowledge base between each test. AIDB can detect changes on volumes even with auto-processing disabled.

Conclusion

In this test, the pipeline took 02:34 min with batch size 1000 and 03:27 min with size 100. You can continue testing larger sizes until performance no longer improves or even declines.

Pipelines knowledge bases usage auto processing

Chosing the auto-processing mode

Also refer to *Pipeline auto-processing* for more background.

You can choose between **Live**, **Background**, and **Disabled** auto processing. The mode can be configured when creating a knowledge base in the `aidb.create_table_knowledge_base` and `aidb.create_volume_knowledge_base` calls.

The mode can also be changed on existing knowledge bases, together with processing settings (like batch size or sync interval) in the `aidb.set_auto_knowledge_base` call.

Prepare the input data

The following examples all use this common source table:

```
CREATE TABLE test_source_table
(
    id            INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    content       TEXT NOT NULL
);
INSERT INTO test_source_table
VALUES (1, Catwalk Women Brown Heels),
       (2, Lakme 3 in 1 Orchid Aqua Shine Lip Color),
       (3, United Colors of Benetton Men Stripes Black Jacket);
```

Using a knowledge base with live auto processing

This processing mode is only supported for table sources since it relies on Postgres Triggers. This mode also does not make use of batch processing since each record is processed individually, immediately when it is created/modified.

Create the knowledge base and configure live auto-processing:

```
SELECT aidb.create_table_knowledge_base(
    name => test_kb_live,
    model_name => bert, -- this is a pre-defined locally running model
    source_table => test_source_table,
    source_data_column => content,
    source_data_format => Text,
    auto_processing => Live
);
__OUTPUT__
INFO:  using vector table: public.test_kb_live_vector
NOTICE: auto-processing is set to "Live". AIDB will process all new/updates rows_
going forward. To process existing data, manually run "SELECT aidb.bulk_
embedding(test_kb_live);"
create_table_knowledge_base
-----
test_kb_live
(1 row)
```

Or manually set an existing knowledge base to live auto-processing:

```
SELECT aadb.set_auto_knowledge_base(test_kb_live, Live);
```

live auto-processing is now in effect for any changes on the source data going forward. But the existing data in the table was not processed yet.

Test this by adding a new record:

```
INSERT INTO test_source_table VALUES (4, This is also a Jacket);
__OUTPUT__
INFO: Running live embedding for knowledge_base test_kb_live. key: "4" content:
↪ "This is also a Jacket"
INSERT 0 1
```

A query will now return the new record, which was auto-processed, but not the pre-existing records:

```
SELECT * FROM aadb.retrieve_text(test_kb_live, Jackets, topk=>4);
__OUTPUT__
key | value | distance
-----+-----+-----
4 | This is also a Jacket | 0.9696326387986498
(1 row)
```

Now run the bulk_embedding call as shown in the NOTICE: above:

```
SELECT aadb.bulk_embedding(test_kb_live);
__OUTPUT__
INFO: test_kb_live: (re)setting state table to process all data...
INFO: test_kb_live: Starting... Batch size 100, unprocessed rows: 4, count(source_
↪ records): 4, count(embeddings): 1
INFO: test_kb_live: Batch iteration finished, unprocessed rows: 0, count(source_
↪ records): 4, count(embeddings): 4
INFO: test_kb_live: finished, unprocessed rows: 0, count(source records): 4,
↪ count(embeddings): 4
bulk_embedding
-----
(1 row)
```

Another retrieval now shows all rows being present:

```
SELECT * FROM aadb.retrieve_text(test_kb_live, Jackets, topk=>4);
__OUTPUT__
key | value | distance
-----+-----+-----
4 | This is also a Jacket | 1.0353689874817347
1 | Catwalk Women Brown Heels | 1.0404924577497399
3 | United Colors of Benetton Men Stripes Black Jacket | 1.324741586641547
2 | Lakme 3 in 1 Orchid Aqua Shine Lip Color | 1.4018234945521992
(4 rows)
```

The statistics view can be queried to see how many records are present, and how many were processed yet:


```
SELECT * FROM aidb.kbstat;
__OUTPUT__
knowledge base | auto processing | table: unprocessed rows | volume: scans completed
↪ | count(source records) | count(embeddings)
-----+-----+-----+-----+
↪ +-----+-----+-----+
test_kb_live   | Live           |                     | 0 |
↪ |               4 |               4      |
(1 row)
```

Using a knowledge base with background auto processing

Background auto-processing is supported on all source types and data types.

Note

Background workers for AIDB must be configured in the Postgres config file; otherwise they will not run. Refer to our installation instructions for more details.

Create the knowledge base and configure background auto-processing. The batch size and sync interval settings are optional.

- short sync intervals lead to shorter delays before embeddings are in sync again. But short delays also cause more processing overhead, especially with volume sources that require (potentially costly) requests to the storage system.
- large batch sizes can lead to higher throughput up to a certain point. This is highly dependent on the infrastructure, data source, type of model, and even the data itself. We recommend users test different batch sizes.

```
SELECT aidb.create_table_knowledge_base(
  name => test_kb_background,
  model_name => bert, -- this is a pre-defined locally running model
  source_table => test_source_table,
  source_data_column => content,
  source_data_format => Text,
  auto_processing => Background,
  batch_size => 50,
  background_sync_interval => 10 seconds
);
__OUTPUT__
INFO:  using vector table: public.test_kb_background_vector
INFO:  test_kb_background: (Re)setting state table to process all data...
NOTICE: auto-processing is set to "Background". Check progress with "SELECT * FROM
↪aidb.kbstat;"
create_table_knowledge_base
-----
test_kb_background
(1 row)
```

Or manually set an existing knowledge base to background auto-processing:

```
SELECT aidb.set_auto_knowledge_base(test_kb_background, Background);
```

Note

When enabling background auto-processing on a pre-existing table knowledge base, only new change events will be processed. If the source table contains unprocessed source data, you can use the `aidb.bulk_embedding()` function to trigger re-processing of all records.

To change the batch size and / or the sync interval:

```
SELECT aidb.set_auto_knowledge_base(test_kb_background, Background, batch_size=>10);
SELECT aidb.set_auto_knowledge_base(test_kb_background, Background, background_sync_
↪interval=>10 minutes);
SELECT aidb.set_auto_knowledge_base(test_kb_background, Background, batch_size=>10, ↪
↪background_sync_interval=>10 minutes);
```

We can now query the statistics like so:

```
SELECT * FROM aidb.kbstat;
__OUTPUT__
knowledge base | auto processing | table: unprocessed rows | volume: scans_
↪completed | count(source records) | count(embeddings)
-----+-----+-----+-----+
↪-----+-----+-----+-----+
test_kb_background | Background | 0 |
↪ | 4 | 4
(1 row)
```

When new data is inserted into the source table, AIDB table triggers will create a change event. This is shown in the table above as `table: unprocessed rows`. When the next `background_sync_interval` is expired, the background worker will process all these changed rows.

We can show this by inserting new records:

```
INSERT INTO test_source_table VALUES (5, This is also another Jacket);
```

and then checking the statistics again:

```
SELECT * FROM aidb.kbstat;
__OUTPUT__
knowledge base | auto processing | table: unprocessed rows | volume: scans_
↪completed | count(source records) | count(embeddings)
-----+-----+-----+-----+
↪-----+-----+-----+-----+
test_kb_background | Background | 1 |
↪ | 5 | 4
(1 row)
```

Using a knowledge base with disabled auto processing

In this mode, no auto-processing happens. Users can manually call `aidb.bulk_embedding()` or enable one of the two auto-processing modes with the `aidb.set_auto_knowledge_base()` call.

AI Accelerator Pipelines - Preparers

Preparers are used to perform common preprocessing steps on source data from either a table or volume source. The processed data is stored in a destination table and can be used by `knowledge_bases` for embedding generation.

Preparers concepts

Preparers are used to perform common preprocessing steps on source data from either a table or volume source. The processed data is stored in a destination table and can be used by other preparers or by `knowledge_bases` for embedding generation.

Concepts

Data preparation operation

The data preparation operation is how the preparer transforms the source data. The supported operations are encoded as variants of the `aidb.DataPreparationOperation` enum.

Note

Each operation has its own set of parameters that are used to customize the operation. Learn more in [Pipelines model primitives](#).

Data sources

A data source is the input data for the data preparation operation. The `aidb` extension supports two types of data sources for preparers:

- Table: a column in a table in the PG database.
- Volume: a PGFS “volume,” which is a wrapper for accessing an S3 object store or local file system.

Execution

Primitive functions help with testing operations and their configurations on individual inputs with minimal setup. This is useful for quick experimentation before scaling up with a preparer for bulk data preparation.

Bulk data preparation performs a preparer’s associated operation for all of the preparer’s source data.

Note

Bulk data preparation doesn't delete existing destination data unless it conflicts with newly generated data. We recommend configuring separate destination tables for each preparer.

Unnesting

Some preparer [Pipelines model primitives](#) transform the shape of the data they are given. For example, `ChunkText` receives one text block and produces one or more text blocks. Rather than return nested collections of results, these primitives unnest (or “explode”) their output, using a new `part_id` column to track the additional dimension.

You can see this in action in [Pipelines model primitives](#) and in the applicable [Where to find SDK references and examples?](#)

Consistency with source data

To ensure correct and consistent data, the prepared destination data must be in sync with the source data. In the case of the table data source, you can enable preparer auto-processing to inform the preparer pipeline about changes to the source data.

Note

If the source table already contains data when the preparer is created, then you must make an initial manual bulk data preparation call.

Preparer auto-processing

You can enable preparer auto-processing to create triggers in Postgres to keep the prepared destination data up to date if source data is added, updated, or removed.

Note

Preparer auto-processing currently works only with preparers with a table data source, not a volume source.

Pipelines data preparation primitives and options

For most use cases, we recommend that you use the aidb *Pipelines knowledge bases usage* to perform bulk preprocessing preparation steps on your data.

However, for testing and developing the configurations that suit your data best, you can use the following primitives. These functions allow you to test operations and their configurations on individual inputs with minimal setup. This is useful for quick experimentation before scaling up with a preparer for bulk data preparation.

Configuration

All data preparation operations can be customized with different options. The API for these options is identical between the primitives and the preparer, so you can prototype options with the `aidb.chunk_text()` primitive for use with a scalable preparer that performs the `ChunkText` operation.

Chunk text

Call `aidb.chunk_text()` to intelligently split long text into smaller, semantically coherent chunks, ideal for processing or storage within LLM context limits.

```
SELECT * FROM aidb.chunk_text(
  input => This is a significantly longer text example that might require splitting
  into smaller chunks. The purpose of this function is to partition text data into
  segments of a specified maximum length, for example, this sentence 145 is
  characters. This enables processing or storage of data in manageable parts.,
  options => {"desired_length": 120, "max_length": 150}
);
```

__OUTPUT__	
part_id	chunk
-----+-----	

(continues on next page)

(continued from previous page)

```

0 | This is a significantly longer text example that might require splitting
↳into smaller chunks.
1 | The purpose of this function is to partition text data into segments of a
↳specified maximum length, for example, this sentence 145 is characters.
2 | This enables processing or storage of data in manageable parts.
(3 rows)

```

Options

- `desired_length` (required): The target chunk size, in characters. This is the size the splitter will try to reach when forming each chunk, while preserving semantic structure. If `max_length` isn't provided, `desired_length` also acts as the hard upper limit for the chunk size.
- `max_length` (optional): The upper bound for chunk size, in characters. If specified, the function will try to generate chunks close to `desired_length` but may extend up to `max_length` to preserve larger semantic units (like full sentences or paragraphs). Chunks will exceed `desired_length` only when it's necessary to avoid cutting across meaningful boundaries.
- Specifying `desired_length = max_length` results in fixed-size chunks (for example, when filling a context window exactly for embeddings).
- Use a larger `max_length` if you want to stay within a soft limit but allow some flexibility to preserve higher semantic units, common in RAG, summarization, or Q&A applications.

Algorithm summary

- Text is split using a hierarchy of semantic boundaries: characters, graphemes, words, sentences, and increasingly long newline sequences (for example, paragraphs).
- The splitter attempts to form the largest semantically valid chunk that fits within the specified size range.
- Chunks may be returned that are shorter than `desired_length` if adding the next semantic unit would exceed `max_length`.

💡 Tip

This operation transforms the shape of the data, automatically unnesting collections by introducing a `part_id` column. See the [unnesting concept](#) for more details.

Parse HTML

Call `aidb.parse_html()` to extract text from HTML.

```

SELECT * FROM aidb.parse_html(
  html =>
    <h1>Hello, world!</h1>
    <p>This is my first web page.</p>
    <p>
      It contains some <strong>bold text</strong>, some <em>italic test</em>,
↳and a <a href="https://google.com" target="_blank">link</a>.
    </p>

```

(continues on next page)

(continued from previous page)

```

<ol>
  <li>List item</li>
  <li>List item</li>
  <li>List item</li>
</ol>,
options => {"method": "StructuredPlaintext"} -- Default
);

__OUTPUT__
          parse_html
-----
Hello, world!                                     +
                                                    +
This is my first web page.                       +
                                                    +
It contains some bold text, some italic test, and a link.+
                                                    +
Postgres Logo Image                             +
List item                                         +
List item                                         +
List item                                         +
(1 row)

```

- The method determines how the HTML is parsed:
- StructuredPlaintext (default) — Algorithmic text extraction to plaintext.
- StructuredMarkdown — Algorithmic text extraction to Markdown-like text that retains some syntactical context.

Parse PDF

Call `aidb.parse_pdf()` to extract text from PDF bytes.

```

SELECT * FROM aidb.parse_pdf(
  bytes => \
→ decode (255044462d312e340a25b89a929d0a312030206f626a3c3c2f547970652f436174616c6f672f506167657320332
→ hex),
  options => {"method": "Structured", "allow_partial_parsing": true} -- Default
);

__OUTPUT__
part_id |      text
-----+-----
      0 | Hello World!+
      |
(1 row)

```

- The method determines how the PDF is parsed:
- Structured (Default) — Algorithmic text extraction. Structural PDF parsing that relies on the PDF specification can struggle with PDFs that are not spec-compliant, especially if they include complex or proprietary

encodings and different fonts.

- The `allow_partial_parsing` flag determines whether to continue to parse PDFs when the parser encounters errors on one or more pages. Defaults to `true`.
- The `part_id` column in the output references the index of the page from which the text was extracted.

Tip

This operation transforms the shape of the data, automatically unnesting collections by introducing a *part_id* column. See the [unnesting concept](#) for more detail.

Summarize text

Call `aidb.summarize_text()` to summarize text:

```
- - Create a model for use in summarization
SELECT aidb.create_model(my_t5_model, t5_local);

SELECT * FROM aidb.summarize_text(
    input => There are times when the night sky glows with bands of color. The bands
    ↳ may begin as cloud shapes and then spread into a great arc across the entire sky.
    ↳ They may fall in folds like a curtain drawn across the heavens. The lights usually
    ↳ grow brighter, then suddenly dim. During this time the sky glows with pale yellow,
    ↳ pink, green, violet, blue, and red. These lights are called the Aurora Borealis.
    ↳ Some people call them the Northern Lights. Scientists have been watching them for
    ↳ hundreds of years. They are not quite sure what causes them. In ancient times Long
    ↳ Beach City College WRSC Page 2 of 2 people were afraid of the Lights. They imagined
    ↳ that they saw fiery dragons in the sky. Some even concluded that the heavens were
    ↳ on fire.,
    options => {"model": "my_t5_model"}
);

__OUTPUT__
create_model
-----
my_t5_model
(1 row)

↳ summarize_text
-----
↳
↳
the night sky glows with bands of color . they may begin as cloud shapes and then
↳ spread into a great arc across the entire sky . the lights usually grow brighter,
↳ then suddenly dim .
(1 row)
```

- The `model` is the name of the created model to use for summarization. The model must support the `decode_text()` and `decode_text_batch()` [Pipelines model primitives](#).

(continued from previous page)

```
);

__OUTPUT__
create_model
- -----
my_paddle_ocr_model
(1 row)

part_id |      text
- -----+-----
      0 | Tesseract sample
(1 row)
```

- The `model` is the name of the created model to use for OCR. The model must support the `perform_ocr` operation.

Tip

This operation transforms the shape of the data, automatically unnesting collections by introducing a `part_id` column. See the [unnesting concept](#) for more detail.

Note

Limitations of the model still apply. For example, the [NVIDIA NIM Image OCR API](#) model provider only supports `png` and `jpeg` image inputs.

Pipelines preparers usage

Creating a new preparer

The source data preparer can come from a Postgres table or a PGFS volume. Given the different nature of the data sources and the options required for each, you use different functions to create them.

Note

You can customize the behavior of the data preparation operation for the preparer with different options. The API for these options is identical between the primitives and the preparer, so you can prototype options with the `aidb.chunk_text()` primitive for use with a scalable preparer that performs the `ChunkText` operation. Learn more in [Pipelines model primitives](#).

Preparer for a table data source

The `aidb.create_table_preparer` function is used to create a preparer for a table data source. This is the function signature:

```
aidb.create_table_preparer(
- -----
name                                TEXT,
```

(continues on next page)

(continued from previous page)

```

operation          aidb.DataPreparationOperation,
source_table       TEXT,
source_data_column TEXT,
destination_table  TEXT,
destination_data_column TEXT,
source_key_column  TEXT DEFAULT id,
destination_key_column TEXT DEFAULT id,
options            JSONB DEFAULT {}::JSONB
)

```

 Tip

The *source_key_column* must be a unique key for the source data. If the data source is the output of a preparer that *transforms the data shape* with a *part_id* column, make sure to use the new *unique_id* column.

Example: Creating a preparer

```

SELECT aidb.create_table_preparer(
  name => test_preparer,
  operation => ChunkText,
  source_table => test_source_table,
  source_data_column => content,
  destination_table => chunked_data_destination_table,
  destination_data_column => chunk,
  source_key_column => id,
  destination_key_column => id,
  options => [{"desired_length": 100}::JSONB -- Configuration for the ChunkText_
  ↳ operation
);

```

Preparer for a volume data source

The *aidb.create_volume_preparer* function is used to create a preparer for a volume data source. This is the function signature:

```

aidb.create_volume_preparer(
-----
  name          TEXT,
  operation      aidb.DataPreparationOperation,
  source_volume_name TEXT,
  destination_table TEXT,
  destination_data_column TEXT,
  destination_key_column TEXT DEFAULT id,
  options        JSONB DEFAULT {}::JSONB
)

```

Example: Creating a preparer for a volume

```
SELECT aidb.create_volume_preparer(
  name => test_preparer_volume,
  operation => ChunkText,
  source_volume_name => test_volume,
  destination_table => chunked_data_destination_table,
  destination_data_column => chunk,
  destination_key_column => id,
  options => [{"desired_length": 100}]::JSONB -- Configuration for the ChunkText_
  ↳operation
);
```

Executing the data preparation

You can use bulk data preparation to execute the processing operation for all existing data in the source location:

```
SELECT aidb.bulk_data_preparation(test_preparer);
```

Enable preparer auto processing to automatically process any future changes to the source data (only for table data sources):

```
SELECT aidb.set_auto_preparer(test_preparer, Live);
```

You can also disable the automatic preparer processing:

```
SELECT aidb.set_auto_preparer(test_preparer, Disabled);
```

Listing the preparers

A view is available that lists all the preparers. *aidb.preparers* also includes some of the preparers configuration:

```
SELECT * FROM aidb.preparers;
__OUTPUT__
id | name | operation | destination_schema | destination_table |
↳| destination_key_column | destination_data_column | options |
↳source_type | source_schema | source_table | source_data_column | source_key_
↳column | source_volume_name
-----+-----+-----+-----+-----+
↳+-----+-----+-----+-----+-----+
↳+-----+-----+-----+-----+-----+
↳+-----+-----+-----+-----+-----+
1 | test_preparer | ChunkText | public | chunked_data_destination_table_
↳| id | | chunk | [{"desired_length": 100}] | Table_
↳| public | test_source_table | content | id |
(1 row)
```

Deleting a preparer

```
SELECT aidb.delete_preparer(<name>);
```

Note

This command doesn't delete the destination table or any data in it.

End to end examples

You can find end-to-end examples for preparers in *Where to find SDK references and examples?* .

AI Accelerator Pipelines - Preparers examples

These examples are grouped by *Data preparation operation* .

Preparer chaining example

Example of chaining multiple preparers together with auto-processing using the *ChunkText* and *SummarizeText* operations in AI Accelerator.

Create the first preparer to chunk text

```
-- Create source test table
CREATE TABLE source_table__1321
(
    id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    content     TEXT NOT NULL
);

SELECT aidb.create_table_preparer(
    name => chunking_preparer__1321,
    operation => ChunkText,
    source_table => source_table__1321,
    source_key_column => id,
    source_data_column => content,
    destination_table => chunked_data__1321,
    destination_data_column => chunk,
    destination_key_column => id,
    options => {"desired_length": 1000}::JSONB -- Configuration for the ChunkText_
--operation
);
```

Create the second preparer to summarize the chunked text

```
-- Create the model. It must support the decode_text and decode_text_batch_
--operations.
SELECT aidb.create_model(model__1321, t5_local);

SELECT aidb.create_table_preparer(
    name => summarizing_preparer__1321,
    operation => SummarizeText,
    source_table => chunked_data__1321,           -- Reference the output from the_
--ChunkText preparer
    source_key_column => unique_id,             -- Reference the unique column from_
```

(continues on next page)

(continued from previous page)

```

-- the output of the ChunkText preparer
source_data_column => chunk,                -- Reference the output from the
-- ChunkText preparer
destination_table => summarized_data__1321,
destination_data_column => summary,
destination_key_column => chunk_unique_id,
options => [{"model": "model__1321"}::JSONB -- Configuration for the
-- SummarizeText operation
);

```

💡 Tip

This operation transforms the shape of the data, automatically unnesting collections by introducing a *part_id* column. See the *unnesting concept* for more detail.

Set both preparers to live auto-processing

```

SELECT aiddb.set_auto_preparer(chunking_preparer__1321, Live);
SELECT aiddb.set_auto_preparer(summarizing_preparer__1321, Live);

```

Insert data for processing

When you insert data into the source data table, you see processed results flowing automatically:

```

INSERT INTO source_table__1321
VALUES (1, This is a significantly longer text example that might require splitting
-- into smaller chunks. The purpose of this function is to partition text data into
-- segments of a specified maximum length, for example, this sentence 145 is
-- characters. This enables processing or storage of data in manageable parts.);

```

Chunks calculated automatically:

```

SELECT * FROM chunked_data__1321;

__OUTPUT__
id | part_id | unique_id |
--
1  | 0       | 1.part.0  | This is a significantly longer text example that might
-- require splitting into smaller chunks.
1  | 1       | 1.part.1  | The purpose of this function is to partition text data
-- into segments of a specified maximum length, for example, this sentence 145 is
-- characters.
1  | 2       | 1.part.2  | This enables processing or storage of data in manageable
-- parts.
(3 rows)

```

Summaries of the chunks calculated automatically:

```
SELECT * FROM summarized_data__1321;
```

```
__OUTPUT__
```

```
chunk_unique_id | summary
```

```
-----+-----
1.part.0      | text example might require splitting into smaller chunks .
1.part.1      | the purpose of this function is to partition text data into
1.part.2      | segments of a specified maximum length .
              | enables processing or storage of data in manageable parts .
(3 rows)
```

The same automatic flow of logic occurs for deletions:

```
DELETE FROM source_table__1321 WHERE id = 1;
```

```
SELECT * FROM chunked_data__1321;
```

```
__OUTPUT__
```

```
id | part_id | unique_id | chunk
```

```
-----+-----+-----+-----
(0 rows)
```

```
SELECT * FROM summarized_data__1321;
```

```
__OUTPUT__
```

```
chunk_unique_id | summary
```

```
-----+-----
(0 rows)
```

Preparers auto-processing example

This example uses preparer auto-processing with the *ChunkText operation* in AI Accelerator.

Tip

This operation transforms the shape of the data, automatically unnesting collections by introducing a *part_id* column. See the [unnesting concept](#) for more detail.

Preparer with table data source

```
-- Create source test table
CREATE TABLE source_table__1628
(
  id      INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  content TEXT NOT NULL
);

SELECT aidb.create_table_preparer(
  name => preparer__1628,
```

(continues on next page)

(continued from previous page)

```

operation => ChunkText,
source_table => source_table__1628,
source_data_column => content,
destination_table => chunked_data__1628,
destination_data_column => chunk,
source_key_column => id,
destination_key_column => id,
options => [{"desired_length": 150}>::JSONB -- Configuration for the ChunkText
->operation
);

SELECT aidb.set_auto_preparer(preparer__1628, Live);

INSERT INTO source_table__1628
VALUES (1, This is a significantly longer text example that might require splitting
->into smaller chunks. The purpose of this function is to partition text data into
->segments of a specified maximum length, for example, this sentence 145 is
->characters. This enables processing or storage of data in manageable parts.);

```

```

SELECT * FROM chunked_data__1628;

__OUTPUT__
id | part_id | unique_id |
->      chunk
-----+-----+-----+-----
1  |      0 | 1.part.0  | This is a significantly longer text example that might
->require splitting into smaller chunks.
1  |      1 | 1.part.1  | The purpose of this function is to partition text data
->into segments of a specified maximum length, for example, this sentence 145 is
->characters.
1  |      2 | 1.part.2  | This enables processing or storage of data in manageable
->parts.
(3 rows)

```

```

INSERT INTO source_table__1628
VALUES (2, This sentence is short enough for desired_length=150 so it wont be chunked.
-> But this second sentence will be in another chunk since both together are over 150
->characters.);

```

```

SELECT * FROM chunked_data__1628;

__OUTPUT__
id | part_id | unique_id |
->      chunk
-----+-----+-----+-----
1  |      0 | 1.part.0  | This is a significantly longer text example that might
->require splitting into smaller chunks.

```

(continues on next page)

(continued from previous page)

```

1 |      1 | 1.part.1 | The purpose of this function is to partition text data_
↪into segments of a specified maximum length, for example, this sentence 145 is_
↪characters.
1 |      2 | 1.part.2 | This enables processing or storage of data in manageable_
↪parts.
2 |      0 | 2.part.0 | This sentence is short enough for desired_length=150 so_
↪it wont be chunked.
2 |      1 | 2.part.1 | But this second sentence will be in another chunk since_
↪both together are over 150 characters.
(5 rows)

```

```
DELETE FROM source_table__1628 WHERE id = 1;
```

```
SELECT * FROM chunked_data__1628;
```

```

__OUTPUT__
id | part_id | unique_id | chunk
---+-----+-----+-----
2 |      0 | 2.part.0 | This sentence is short enough for desired_length=150 so_
↪it wont be chunked.
2 |      1 | 2.part.1 | But this second sentence will be in another chunk since_
↪both together are over 150 characters.
(2 rows)

```

```
SELECT aidb.set_auto_preparer(preparer__1628, Disabled);
```

Preparers chunk text operation examples

These examples use preparers with the *ChunkText operation* in AI Accelerator.

Tip

This operation transforms the shape of the data, automatically unnesting collections by introducing a *part_id* column. See the [unnesting concept](#) for more detail.

Primitive

```

-- Only specify a desired length
SELECT * FROM aidb.chunk_text(This is a simple test sentence., {"desired_length":_
↪10});

__OUTPUT__
part_id | chunk
---+-----
0 | This is a
1 | simple
2 | test
3 | sentence.
(4 rows)

```

```
-- Specify a desired length and a maximum length
SELECT * FROM aidb.chunk_text(This is a simple test sentence., {"desired_length": 10,
  ↳"max_length": 15});
```

```
__OUTPUT__
part_id | chunk
-----+-----
      0 | This is a
      1 | simple test
      2 | sentence.
(3 rows)
```

```
-- Named parameters
SELECT * FROM aidb.chunk_text(
  input => This is a significantly longer text example that might require splitting.
  ↳into smaller chunks. The purpose of this function is to partition text data into
  ↳segments of a specified maximum length, for example, this sentence 145 is
  ↳characters. This enables processing or storage of data in manageable parts.,
  options => {"desired_length": 40}
);
```

```
__OUTPUT__
part_id | chunk
-----+-----
      0 | This is a significantly longer text
      1 | example that might require splitting
      2 | into smaller chunks.
      3 | The purpose of this function is to
      4 | partition text data into segments of a
      5 | specified maximum length, for example,
      6 | this sentence 145 is characters.
      7 | This enables processing or storage of
      8 | data in manageable parts.
(9 rows)
```

Preparer with table data source

```
-- Create source test table
CREATE TABLE source_table__1628
(
  id      INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  content TEXT NOT NULL
);
INSERT INTO source_table__1628
VALUES (1, This is a significantly longer text example that might require splitting.
  ↳into smaller chunks. The purpose of this function is to partition text data into
  ↳segments of a specified maximum length, for example, this sentence 145 is
  ↳characters. This enables processing or storage of data in manageable parts.),
      (2, This is sentence number one. This is sentence number one.);

SELECT aidb.create_table_preparer(
  name => preparer__1628,
```

(continues on next page)

```

operation => ChunkText,
source_table => source_table__1628,
source_data_column => content,
destination_table => chunked_data__1628,
destination_data_column => chunks,
source_key_column => id,
destination_key_column => id,
options => {"desired_length": 120}::JSONB -- Configuration for the ChunkText
->operation
);

SELECT aidb.bulk_data_preparation(preparer__1628);

SELECT * FROM chunked_data__1628;

__OUTPUT__
id | part_id | unique_id |
->chunks
- - - - -
1 | 0 | 1.part.0 | This is a significantly longer text example that might
->require splitting into smaller chunks.
1 | 1 | 1.part.1 | The purpose of this function is to partition text data
->into segments of a specified maximum length, for example, this
1 | 2 | 1.part.2 | sentence 145 is characters. This enables processing or
->storage of data in manageable parts.
2 | 0 | 2.part.0 | This is sentence number one. This is sentence number one.
(4 rows)

```

```
- - Default method is to structurally parse HTML to plaintext
SELECT * FROM aidb.parse_html(
    <html><body><h1>Hello World Heading</h1><p>Hello World paragraph</p></body></html>
);

__OUTPUT__
      parse_html
-----
Hello World Heading  +
                    +
Hello World paragraph+

(1 row)
```

```
- - Parse Hello World HTML to plaintext
SELECT * FROM aidb.parse_html(
```

(continued from previous page)

```

html =>
  <h1>Hello, world!</h1>
  <p>This is my first web page.</p>
  <p>
    It contains some <strong>bold text</strong>, some <em>italic test</em>,
    and a <a href="https://google.com" target="_blank">link</a>.
  </p>

  <ol>
    <li>List item</li>
    <li>List item</li>
    <li>List item</li>
  </ol>,
  options => [{"method": "StructuredPlaintext"}] -- Default
);

```

__OUTPUT__

parse_html

```

-----
Hello, world!                                     +
                                                    +
This is my first web page.                         +
                                                    +
It contains some bold text, some italic test, and a link.+
                                                    +
Postgres Logo Image                             +
List item                                       +
List item                                       +
List item                                       +

```

(1 row)

```

-- Parse Hello World HTML to markdown-esque text that retains some syntactical
context
SELECT * FROM aidb.parse_html(
  html =>
    <h1>Hello, world!</h1>
    <p>This is my first web page.</p>
    <p>
      It contains some <strong>bold text</strong>, some <em>italic test</em>,
      and a <a href="https://google.com" target="_blank">link</a>.
    </p>

    <ol>
      <li>List item</li>
      <li>List item</li>
      <li>List item</li>
    </ol>,

```

(continues on next page)

(continued from previous page)

```

    options => {"method": "StructuredMarkdown"}
);

__OUTPUT__

                                parse_html
-----
↪ --
# Hello, world!
↪ +
↪ +
This is my first web page.
↪ +
↪ +
It contains some bold text, some italic test, and a [link] (https://google.com).
↪ +
↪ +
![(Postgres Logo Image) (postgres_logo.png)]
↪ +
1. List item
↪ +
2. List item
↪ +
3. List item
↪ +
(1 row)

```

Preparer with table data source

```

-- Create source test table
CREATE TABLE source_table__2772
(
    id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    content     TEXT NOT NULL
);
INSERT INTO source_table__2772
VALUES (1, <html><body><h1>Hello World Heading</h1><p>Hello World paragraph</p></body>
↪ </html>),
      (2, <p>This is some <strong>bold text</strong>, some <em>italic test</em>, and
↪ a <a href="https://google.com" target="_blank">link</a>.);

SELECT aidb.create_table_preparer(
    name => preparer__2772,
    operation => ParseHtml,
    source_table => source_table__2772,
    source_data_column => content,
    destination_table => destination_table__2772,
    destination_data_column => parsed_html,

```

(continues on next page)

(continued from previous page)

```

    source_key_column => id,
    destination_key_column => id,
    options => [{"method": "StructuredPlaintext"}>::JSONB -- Configuration for the
↳ ParseHtml operation
);

SELECT aidb.bulk_data_preparation(preparer__2772);

SELECT * FROM destination_table__2772;

__OUTPUT__
id |                parsed_html
-----+-----
1  | Hello World Heading          +
    |                             +
    | Hello World paragraph       +
2  | This is some bold text, some italic test, and a link.+
    |
(2 rows)

```

Preparers parse PDF operation examples

These examples use preparers with the *ParsePdf operation* in AI Accelerator.

Tip

This operation transforms the shape of the data, automatically unnesting collections by introducing a *part_id* column. See the *unnesting concept* for more detail.

Primitive

```

-- Parse Hello World PDF with default options
-- Bytes can come from a binary column or a manual `pg_read_binary_file()` call
SELECT * FROM aidb.parse_pdf(
  de-
code(255044462d312e340a25b89a929d0a312030206f626a3c3c2f547970652f436174616c6f672f50616765732033203020
↳ hex)
);

__OUTPUT__
part_id |      text
-----+-----
0  | Hello World!+
    |
(1 row)

```

```

-- Manually specify the default options
SELECT * FROM aidb.parse_pdf(
  bytes =>

```

(continues on next page)

(continued from previous page)

```

→ decode (255044462d312e340a25b89a929d0a312030206f626a3c3c2f547970652f436174616c6f672f5061676573203320
→ hex),
    options => {"method": "Structured", "allow_partial_parsing": true} -- Default
);

__OUTPUT__
part_id |      text
-----+-----
      0 | Hello World!+
      |
(1 row)

```

Preparer with table data source

```

-- Create source test table
CREATE TABLE source_table__6124
(
    id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    content     TEXT NOT NULL
);
INSERT INTO source_table__6124
VALUES (1, →
→ decode (255044462d312e340a25b89a929d0a312030206f626a3c3c2f547970652f436174616c6f672f5061676573203320
→ hex)),
      (2, →
→ decode (255044462d312e340a25b89a929d0a312030206f626a3c3c2f547970652f436174616c6f672f5061676573203320
→ hex));

SELECT aedb.create_table_preparer(
    name => preparer__6124,
    operation => ParsePdf,
    source_table => source_table__6124,
    source_data_column => content,
    destination_table => destination_table__6124,
    destination_data_column => parsed_pdf,
    source_key_column => id,
    destination_key_column => id,
    options => {"method": "Structured"}::JSONB -- Configuration for the ParsePdf_
→ operation
);

SELECT aedb.bulk_data_preparation(preparer__6124);

SELECT * FROM destination_table__6124;

__OUTPUT__
id | part_id | unique_id |  parsed_pdf
---+-----+-----+-----
 1 |      0 | 1.part.0 | Hello World!+
  |      |  |
 2 |      0 | 2.part.0 | Hello World!+

```

(continues on next page)

Preparers perform OCR operation examples

 **Tip**

This operation transforms the shape of the data, automatically unnesting collections by introducing a *part_id* column. See the *unnesting concept* for more detail.

This step is required for primitive single execution and for preparer bulk execution.

```
- - Create the model. It must support the perform_ocr operation.
SELECT aidb.create_model(
    my_paddle_ocr_model,
    nim_paddle_ocr,
    credentials => { "api_key": "__NVIDIA_NIM_API_KEY__" }::JSONB
);
```

[illegible]

307

[illegible]

(continues on next page)

```

→ fouj15JY6RmYqNsUFPEyKJjfAMaLgmAPkpKkHMFC1jTz0Eh9LzI0PJTbJbaXVhXtP1TA+FlqVtQ5h9dy8vQAA+waEiiAqrOqY8L
→ dkieXsoa26L+ML83aX+op5EFLRh4Z7ueux1YI6WBkZLZWA0cux9ZdUxEMe6RhDH+j1J/
→ RTUrJAzN1HNs49PqfSzaq7IV+RqFFj1jiE7GjzExlWXST15eJqV84F7m+BYF2Ss4Aeu7X/
→ AA6pKyYVrIGVceHdiVgTSsDID3gPAbgmCsChqmTRh7JWxgEh3qCFVPjZVzPYaR5xFuEediop7Q3VUaIgPuOEOLa4kqw6fFE8xTU
→ ACFVvmbuZKyYF9zp34gMK7OWdTxXc8vMTSAPcl6LDRO7RWZFCWC5pEL2RphLKGuFHNwE7fE5FOzv1kySMFET950T3l7XBVLiaar
→ sbjndtsGSnAr5qE1M8jTGHMcWaldmaXfP/
→ tYRunDiCFakr2vibVRfiwEOiCkKjFtUMVTW85ETDhVBfANbWY33Sm9kVzixWdRUs9ZSvpaaNkTQ8umC/
→ Eouj17UOz8rVdWL+6qP/wBTV/eY/
→ wBDl+9OjdTC1ayKd3tHM8sJUzJYZWB0b2G80dvUFTtdXV08b3RjzMiJUZYa6te9vM2ML83aX+oqAiKamfC9/
→ GNdm7Nkl3EbKhpyC4StFxxhdlrOENMmwuwlGbf7MHERsPFY/
→ dY2MdUAg4yf1PqObZx6fUgGSEkmjYD4XuaQFSQRUndZo8TJg43yJoc1zS1zT4EO9FUhoMpl7sX7p8TuQqrteU7xc8PnAap2VNr
→ Rgg3QH6wP4l6tEzRknBLTS7p/8TSqmsFK/
→ z94nDI10Ki16pobLMBcxjP3GKZlLbEEeASHySNHgX6qa1tMz7g7vUB0StAiEPbKy6XeTTAfbbhVowSSUHD6WLyTG64KmiJdVPh
→ zDVXmojB04mil3c4BVVUmmxjGKmoAYFM2ptKrINVVe4Hgxns0KoZFX0VRv4d55X/
→ dIc0lVZhsmtGCSohZV3s3QlBftnigtGX1MDzgErX9gcFU2juQLmXtTcq4iLGHYxbzezy+zT7KRlLV0Lg+iP6jcIuwkexCNZD
→ PUTnzSv8AqObZx6ftQf8AX9nJhwie1P2BzbOPT4JQwRyG++RsbQ84uLR+wObZx6fJObZx6fJObZx6fJObZx6fJObZx6fJObZx6f
→ p8mGz//xAAUEQEAAAAAAAAAAAAAAAAAACg/9oACAECAQE/ABK//8QAFBEBAAAAAAAAAAAAAAAAAAAAAoP/
→ aaAgBAwEBPwASv//Z, base64),
options => {"model": "my_paddle_ocr_model"}
);

__OUTPUT__
part_id |      text
-----+-----
0 | Tesseract sample
(1 row)

```

[illegible]

[illegible]

(continues on next page)

```

→ fouj15JY6RmYqNsUFPEyKJjfAMaLgmAPkpKkHMFC1jTz0Eh9LzI0PJTbjbaXVhXtP1TA+FlqVtQ5h9dy8vQAA+waEiiAqrOqY8L
→ dkieXsoa26L+ML83aX+op5EFLRh4Z7ueux1YI6WBkZLZWA0cux9ZdUxEMe6RhDH+j1J/
→ RTUrJAzNHNs49PqfSzaq7IV+RqFFfjljiE7GjzExlWXST15eJqV84F7m+BYF2Ss4Aeu7X/
→ AA6pKyYVrIGVceHdiVgTSsDID3gPAbgmCsChqmTrh7JWxgEh3qCFVPjzVzPYaR5xFuEediop7Q3VUaIgPuOEOLa4kqw6fFE8xTU
→ ACFVvmbuZKyYF9zp34gMK7OWdTxXc8vMTSAPcl6LDRO7RWZFCWC5pEL2RphLKGuFHNwE7fE5FOzv1kySMfET950T3l7XBVLiaar
→ sbjndtsGSnAr5qE1M8jTGHMcWaldmaXfP/
→ tYRunDiCFakr2vibVRfiwEOiCkKjFtUMVTW85ETDhVBfANbWY33Sm9kVzixWdRUs9ZSvpaaNkTQ8umC/
→ Eouj17UOz8rVdWL+6qP/wBTV/eY/
→ wBDl+9OjdTC1ayKd3tHM8sJUzJYZWB0b2G80dvUFTtdXV08b3RjzMiJUZYa6te9vM2ML83aX+oqAiKamfC9/
→ GNdm7Nkl3EbKhpyC4StFxxhdlrOENMmwuwlGbf7MHERsPFY/
→ dY2MdUAg4yf1PqObZx6fUgGSEkmjYD4XuaQFSQRUndZo8TJg43yJoc1zS1zT4EO9FUhoMpl7sX7p8TuQqrteU7xc8PnAap2VNr
→ Rgg3QH6wP4l6tEzRknBLTS7p/8TSqmsFK/
→ z94nDI10Ki16pobLMBcxjP3GKZlLbEEeASHySNHgX6qa1tMz7g7vUB0StAiEPbKy6XeTTAfbbhVowSSUHD6WLyTG64KmiJdVPh
→ zDVXmojB04mil3c4BvVUmmxjGKmoAYFM2ptKrINVVe4Hgxns0KoZFX0VRv4d55X/
→ dIc0lVZhsmtGCSohZV3s3QlBftnigtGX1MDzgErX9gcFU2juQLmXtTcq4iLGHYxbzezy+zT7KRlLV0Lg+iP6jcIuwkexCNZD
→ PUTnzSv8AqObZx6ftQf8AX9nJhwie1P2BzbOPT4JQwRyG++RsbQ84uLR+wObZx6fJObZx6fJObZx6fJObZx6fJObZx6fJObZx6f
→ p8mGz//xAAUEQEAAAAAAAAAAAAAAAAAACg/9oACAECAQE/ABK//8QAFBEBAAAAAAAAAAAAAAAAAAAAAoP/
→ aAAgBAwEBPwASv//Z, base64),
{"model": "my_paddle_ocr_model"}
);

__OUTPUT__
part_id |      text
-----+-----
0 | Tesseract sample
(1 row)

```

[illegible]

[illegible]

(continues on next page)

(continued from previous page)

```

→M2n1KaJJvJTQeBkkKrJ4qMO/Vl7rTt4MKral1KXAXul7xTycpKYI52nBVQfhYj6jm2cen1PpZtVdkK/
→I1CfhfUGOnBBuID1YE9bXVTgGzxva3BEB5V2Rr/AOexUT7PgmtOCUQuIJbicA//
→ACKpH1MDKQR1LGeZmFWJ3tsDBGJYzuprmqIxVsbcEIggY/MfKHBNZHuoIu4sLvu/
→0P2gf5hW0dzLLfUUtQwt++BdiY5WTVUsFVHupmkCVjghA6mrLt5JE8uxYfAG9UcldTxWnNO6naQC7CTgAv8AQFdkK7+exdn56O
→sxktzQoWyiyoY9yx/
→4s5P3tloystgGR4iMxneDN5jgVi11W7wYTdEFSimnr00FBIYgCAwb5gATiGVtdLLIPfu7P/
→siiHebUlmqJ3n2a8xsGUJrXteC1zXC8EO9CpKwTtbIwMfJezDIF+bqV+S/
→3O20sktHNVx1kFREMe5mFxcuzpMgAa6ekIGZpTGPtt+7O7nvhe8xqR7KCGQVDnMl3IaYwftJVXU1UMc75SIgZcTj7vK7PTMiqr
→fouj15JY6RmYqNsUFPEyKJjfAmaLgmAPkpKkHMFC1jTZ0Eh9LzI0PJTbJbaXVhXtP1TA+FlqVtQ5h9dy8vQAA+wAeiiAqrOqY8I
→dkieXsoa26L+ML83aX+op5EFLRh4Z7ueux1YI6WBkZLZWAocux9ZdUxEMe6RhDH+j1J/
→RTUrJAzN1HNs49PqfSzaq7IV+RqFFj1jE7GjzExlWXST15eJqV84F7m+BYF2Ss4Aeu7X/
→AA6pKyYVrIGVceHdiVgTSsDID3gPabgmCsChqmTRh7JWxgEh3qCFVPjZVzPYaR5xFuEediop7Q3VUaIgpUOEoiA4kqw6fFE8xT
→ACFVvmbuZKyYf9zp34gMK7OWdTxxC8vMTSAPcl6LDRO7RWZFCWC5pEL2RphLKGuFHNwE7fE5FOzv1kySMfET950T3l7XBVLiaar
→sbjndtsGSnAr5qE1M8jTGHMcWaldmaXfP/
→tYRunDiCFakr2vibVRfiwEOICkkjFtUMVTW85ETDhVBFaNBWY33Sm9kVzixWdRUs9ZSvpaaNkTQ8umC/
→Eouj17UOz8rVdWL+6qP/wBTv/eY/
→wBDl+9OjdTC1ayKd3tHM8sJUzJYZWB0b2G8ODvUFTtdXV08b3RjzMiJUZYa6te9vM2ML83aX+oqAiKamfC9/
→GNdm7Nkl3EbKhpYC4StFxxhdlrOENMwuwlgBf7MHersPFY/
→dY2MdUAg4yf1PqObZx6fUgGsekmjYD4XuaQFSQRUndZo8TJg43yJoc1zS1zT4EO9FUhoMpl7sX7p8TuQqrteU7xc8PnAap2VNr
→Rgg3QH6wP4l6tEzRknBLTS7p/8TSqmsFK/
→z94nDI10Ki16pobLMBcxjP3GKZlLbEEeASHySNHgx6qa1tMz7g7vUB0StAiEPBky6XeTTAfbbhVowSSUhd6WLyTGG64KmiJdVPh
→zDVXmojBO4mil3c4BVVUmmxjGKmoAYFM2ptKrINVVe4Hgxns0KoZFX0VRv4d55X/
→dIc0lVZhsmtGCSohZV3s3QlBftnigtGQX1MDzgErX9gcFU2juQLmXTtcq4iLGHYxbzezy+zT7KRlLV0Lg+iP6jcIuwkexCNZD
→PUTnzSv8AQobZx6ftQf8AX9nJhwie1P2BzbOPT4JQwRyG++RsbQ84uLR+wObZx6fJObZx6fJObZx6fJObZx6fJObZx6fJObZx6f
→p8mGz//xAAUEQEAAAAAAAAAAAAAAAAAACg/9oACAECAQE/ABK//8QAFBEBAAAAAAAAAAAAAAAAAAAAoP/
→aAAgBAwEBPwASv//Z, base64));

```

```

SELECT aidb.create_table_preparer(
  name => preparer__1527,
  operation => PerformOcr,
  source_table => source_table__1527,
  source_data_column => content,
  destination_table => ocr_data__1527,
  destination_data_column => parsed__text,
  source_key_column => id,
  destination_key_column => id,
  options => [{"model": "model__1527"}::JSONB -- Configuration for the PerformOcr_
→operation
);

```

```
SELECT aidb.bulk_data_preparation(preparer__1527);
```

```
SELECT * FROM ocr_data__1527;
```

__OUTPUT__			
id	part_id	unique_id	parsed__text
1	0	1.part.0	Trunch Parish Council
1	1	1.part.1	BANK RECONCILIATION AS AT 31STOCTOBER 2019
1	2	1.part.2	Account:
1	3	1.part.3	14,389.43
1	4	1.part.4	BANK STATEMENT BALANCE 30TH SEPTEMBER 2019

(continues on next page)

1		5	1.part.5	83.60
1		6	1.part.6	PREVIOUS OUTSTANDING CHEQUES
1		7	1.part.7	14,305.83
1		8	1.part.8	CASHBOOK BALANCE 31ST OCTOBER 2019
1		9	1.part.9	ADD CHEQUES OUTSTANDING:
1		10	1.part.10	*
1		11	1.part.11	101719
1		12	1.part.12	83.60*
1		13	1.part.13	*
1		14	1.part.14	*
1		15	1.part.15	83.60
1		16	1.part.16	OUTSTANDING CHEQUES
1		17	1.part.17	9,148.00
1		18	1.part.18	RECEIPTS
1		19	1.part.19	4,309.94
1		20	1.part.20	PAYMENTS
1		21	1.part.21	19,227.49
1		22	1.part.22	BALANCE 31STOCTOBER2019
1		23	1.part.23	19,227.49*
1		24	1.part.24	BALANCE AS PER BANK STATEMENT
1		25	1.part.25	0.00
1		26	1.part.26	DIFFERENCE
(27 rows)				

[illegible]

(continues on next page)

(continued from previous page)

```

→ACKpH1MDKQR1LGeZmFWJ3tsDBGJYzuprmqIxVsbcEIggY/MfKHBNZHuoIu4sLvu/
→0P2gf5hW0dzLLfUUUQwt++BdiY5WTUVsFVHupmkCVjghA6mrLt5JE8uxYfAG9UcldTxWnNO6naQC7CTgAv8AQFdkK7+exdn56Ov
→sxktzQoWyiyoy9yx/
→4s5P3tloystgGR4iMxneDN5jgVi11W7wYTdEFSimnr00FBIYgCAwb5gATiGVtdLLIPfu7P/
→siiHebUlmqJ3n2a8xsGUJrXteC1zXC8EO9CpKwTtbIwMfJezDIF+bqV+S/
→3O20sktHNVx1kFREMe5mFxcuzpMgAa6ekIGZpTGPtt+7O7nvhe8xqR7KCGQVDnMl3IaYwftJVXU1UMc75SIgZcTj7vK7PTMiqr
→fouj15JY6RmYqNsUFPEyKJjfAMaLgmAPkpKkHMFC1jTZ0Eh9LzI0PJTbJbaXVhXtP1TA+FlqVtQ5h9dy8vQAA+wAeiiAqrOqY8I
→dkieXsoa26L+ML83aX+op5EFLRh4Z7ueux1YI6WBkZLZWA0cux9ZdUxEMe6RhDH+j1J/
→RTUrJAzN1HNs49PqfSzaq7IV+RqFFj1jie7GjzExlWXST15eJqV84F7m+BYF2Ss4Aeu7X/
→AA6pKyYVrIGVceHdiVgTSsDID3gPabgmCsChqmTRh7JWxgEh3qCFVPjZVzPYaR5xFuEediop7Q3VUaIgPuOE0Ia4kqw6fFE8xT
→ACFVvmbuZKyYF9zp34gMK7OWdTxxC8vMTSAPcl6LDRO7RWZFCWC5pEL2RphLKGuFHNwE7fE5FOzv1kySMfET950T3l7XBVLiaar
→sbjndtsGSnAr5qE1M8jTGHMcWaldmaXfP/
→tYRunDiCFakr2vibVRfiwEOICkkjFtUMVTW85ETDhVBFaNBWY33Sm9kVzixWdRUs9ZSvpaaNkTQ8umC/
→Eouj17UOz8rVdWL+6qP/wBTv/eY/
→wBDl+9OjdTC1ayKd3tHM8sJUzJYZWB0b2G8ODvUFTtdXV08b3RjzMiJUZYa6te9vM2ML83aX+oqAiKamfC9/
→GNdm7Nkl3EbKhpYC4StFxxhdlrOENMwuwlgBf7MHersPFY/
→dY2MdUAG4yf1pQObZx6fUgGsekjYD4XuaQFSQRUndZo8TJg43yJoc1zS1zT4EO9FUhoMpl7sX7p8TuQqrteU7xc8PnAap2VNr
→Rgg3QH6wP4l6tEzRknBLTS7p/8TSqmsFK/
→z94nDI1OKi16pobLMBcxjP3GKZlLbEEeASHySNHgx6qa1tMz7g7vUB0StAiEPBkY6XeTTAfbbhVowSSUhd6WLyTGG64KmiJdVPh
→zDVXmojBO4mil3c4BVVUmmxjGKmoAYFM2ptKrINVVe4HgXns0KoZFX0VRv4d55X/
→dIc0lVZhsmtGCSohZV3s3QlBftnigtGQX1MDzgErX9gcFU2juQLmXTtcq4iLGHYxbzezy+zT7KRlLV0Lg+iP6jcIuwkexCNZD
→PUTnzSv8AQObZx6ftQf8AX9nJhwie1P2BzbOPT4JQwRyG++RsbQ84uLR+wObZx6fJObZx6fJObZx6fJObZx6fJObZx6fJObZx6f
→p8mGz//xAUEQEAAAAAAAAAAAAAAAAAACg/9oACAECAQE/ABK//8QAFBEBAAAAAAAAAAAAAAAAAAAAoP/
→aAAgBAwEBPwASv//Z, base64),
    options => {"model": "bert_model"}
);
__OUTPUT__
ERROR: The requested adapter is not supported by the model provider: bert_local

```

```

- - Preparer creation fails
SELECT aidb.create_table_preparer(
  name => preparer__1527,
  operation => PerformOcr,
  source_table => source_table__1527,
  source_data_column => content,
  destination_table => ocr_data__1527,
  destination_data_column => parsed__text,
  source_key_column => id,
  destination_key_column => id,
  options => {"model": "bert_model"}::JSONB -- Incompatible model
);
__OUTPUT__
ERROR: Failed to create preparer: The requested adapter is not supported by the
→model provider: bert_local

```

Preparers summarize text operation examples

Examples of using preparers with the *SummarizeText operation* in AI Accelerator.

Model creation (required)

This step is required for primitive single execution and for preparer bulk execution.

```
- - Create the model. It must support the decode_text and decode_text_batch
  ↳ operations.
SELECT aidb.create_model(model__1952, t5_local);
```

Primitive

```
- - Keyword arguments
SELECT * FROM aidb.summarize_text(
    input => I dont want a babysitter. I am eleven years old. My babysitter is only
  ↳ three years older than I am," she loudly yelled to her Mom. Now, she really wished
  ↳ she had somebody with her as she heard the clicking, scratching noises outside of
  ↳ the living room window. "This is silly. Its probably the storm," the girl said. She
  ↳ regretted watching the horror show she had been tuned into for the last half hour.
  ↳ As she searched for the remote to turn off the vampire movie, the front door blew
  ↳ open with a thunderous noise. Carla whirled around to see a dark image.,
    options => {"model": "model__1952"}
);

__OUTPUT__

  ↳ summarize_text

-----

the girl yelled to her Mom as she heard the clicking, scratching noises outside of
  ↳ the living room window . she regretted watching the horror show she had been tuned
  ↳ into for the last half hour . the front door blew open with a thunderous noise .
(1 row)
```

```
- - Positional arguments
SELECT * FROM aidb.summarize_text(
    There are times when the night sky glows with bands of color. The bands may begin
  ↳ as cloud shapes and then spread into a great arc across the entire sky. They may
  ↳ fall in folds like a curtain drawn across the heavens. The lights usually grow
  ↳ brighter, then suddenly dim. During this time the sky glows with pale yellow, pink,
  ↳ green, violet, blue, and red. These lights are called the Aurora Borealis. Some
  ↳ people call them the Northern Lights. Scientists have been watching them for
  ↳ hundreds of years. They are not quite sure what causes them. In ancient times Long
  ↳ Beach City College WRSC Page 2 of 2 people were afraid of the Lights. They imagined
  ↳ that they saw fiery dragons in the sky. Some even concluded that the heavens were
  ↳ on fire.,
    {"model": "model__1952"}
);

__OUTPUT__

  ↳ summarize_text

-----
```

(continues on next page)

(continued from previous page)

```

→-----
the night sky glows with bands of color . they may begin as cloud shapes and then
→spread into a great arc across the entire sky . the lights usually grow brighter,
→then suddenly dim .
(1 row)

```

Preparer with table data source

```

-- Create source test table
CREATE TABLE source_table__1952
(
  id      INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  content TEXT NOT NULL
);
INSERT INTO source_table__1952
VALUES (1, I dont want a babysitter. I am eleven years old. My babysitter is only
→three years older than I am," she loudly yelled to her Mom. Now, she really wished
→she had somebody with her as she heard the clicking, scratching noises outside of
→the living room window. "This is silly. Its probably the storm," the girl said. She
→regretted watching the horror show she had been tuned into for the last half hour.
→As she searched for the remote to turn off the vampire movie, the front door blew
→open with a thunderous noise. Carla whirled around to see a dark image.),
      (2, There are times when the night sky glows with bands of color. The bands
→may begin as cloud shapes and then spread into a great arc across the entire sky.
→They may fall in folds like a curtain drawn across the heavens. The lights usually
→grow brighter, then suddenly dim. During this time the sky glows with pale yellow,
→pink, green, violet, blue, and red. These lights are called the Aurora Borealis.
→Some people call them the Northern Lights. Scientists have been watching them for
→hundreds of years. They are not quite sure what causes them. In ancient times Long
→Beach City College WRSC Page 2 of 2 people were afraid of the Lights. They imagined
→that they saw fiery dragons in the sky. Some even concluded that the heavens were
→on fire.);

SELECT aidb.create_table_preparer(
  name => preparer__1952,
  operation => SummarizeText,
  source_table => source_table__1952,
  source_data_column => content,
  destination_table => summarized_data__1952,
  destination_data_column => summary,
  source_key_column => id,
  destination_key_column => id,
  options => {"model": "model__1952"}::JSONB -- Configuration for the
→SummarizeText operation
);

SELECT aidb.bulk_data_preparation(preparer__1952);

SELECT * FROM summarized_data__1952;

__OUTPUT__

```

(continues on next page)

(continued from previous page)

```

id |
   |
   | summary
   |-----+-----
   |-----+-----
   |-----+-----
1  | the girl yelled to her Mom as she heard the clicking, scratching noises outside
   | of the living room window . she regretted watching the horror show she had been
   | tuned into for the last half hour . the front door blew open with a thunderous
   | noise .
2  | the night sky glows with bands of color . they may begin as cloud shapes and
   | then spread into a great arc across the entire sky . the lights usually grow
   | brighter, then suddenly dim .
(2 rows)

```

Model compatibility

The AI accelerator data preparation pipeline validates the options at creation time, including the model compatibility with the language adapter. These errors are returned in that case:

```

-- This model does not support the language adapter
SELECT aidb.create_model(bert_model, bert_local);

```

```

-- Single execution fails
SELECT * FROM aidb.summarize_text(
    input => Hello world,
    options => {"model": "bert_model"}
);
__OUTPUT__
ERROR:  The requested adapter is not supported by the model provider: bert_local

```

```

-- Preparer creation fails
SELECT aidb.create_table_preparer(
    name => preparer_name,
    operation => SummarizeText,
    source_table => source_table__1952,
    source_data_column => content,
    destination_table => summarized_data__1952,
    destination_data_column => summary,
    options => {"model": "bert_model"}::JSONB -- Incompatible model
);
__OUTPUT__
ERROR:  Failed to create preparer: The requested adapter is not supported by the
model provider: bert_local

```

Pipelines data preparation output to volumes

The output of a preparer pipeline can be stored in a Postgres table or in an AIDB volume (with PGFS storage location). Most examples show how to use tables; this page describes how you can store the output in a volume.

Output format

Each result will be stored in its own object. The name/key of the object will be the same name/key that the source record has. Operations that produce lists of results, like chunking, OCR, or HTML parsing, will store each “part” in its own object.

E.g., a single object `long_text.txt` will be split into chunks; each chunk is stored a separate object. They will have the following names: `long_text.txt.part.0`, `long_text.txt.part.1`, and so on.

Preparing a destination volume

A PGFS storage location and an AIDB volume for the intended destination are needed. The volume needs to be writable; whether it is an S3 bucket or a local directory.

Also refer to the :ref: `AIDB volumes for accessing PGFS storage locations` <AIDB volumes for accessing PGFS storage locations>` and :ref: `Pipelines PGFS reference` <Pipelines PGFS reference>` docs for more details.

Here, we use a local directory (on the Postgres server) as the destination.

Prepare a local directory:

```
mkdir -p /tmp/pgfs/preparer_output_chunks
```

Set up PGFS, allow access to the directory, and configure the storage location:

```
CREATE EXTENSION IF NOT EXISTS pgfs;
SET pgfs.allowed_local_fs_paths = /tmp/pgfs;

SELECT pgfs.create_storage_location(local_tmp_pgfs, file:///tmp/pgfs/);
```

Now set up an AIDB volume for this storage location. Optionally configure the sub-path (here `preparer_output_chunks`). Make sure to configure the correct data format; in this case `Text` since we will use a text chunking preparer pipeline.

```
SELECT aidb.create_volume(chunk_output_volume, local_tmp_pgfs, preparer_output_chunks/
↪, Text);
```

Test whether the volume can be accessed and confirm that it is empty. Pipelines will write to non-empty destinations with the risk of overwriting existing data.

```
SELECT * FROM aidb.list_volume_content(chunk_output_volume);
```

Preparing source data

Volumes and tables can both be used as a source; the destination type doesn't affect this. Here we use a volume as a realistic example.

```
SELECT pgfs.create_storage_location(
    text_files_bucket_aidb, s3://aidb-rag-app-txt,
    options => {"region": "eu-central-1", "skip_signature": "true"});
SELECT aidb.create_volume(text_files_volume, text_files_bucket_aidb, /, Text);

SELECT * FROM aidb.list_volume_content(text_files_volume);
```

Creating the preparer pipeline and run it

Use the `aidb.create_volume_preparer()` call passing the source and destination volumes we just created.

Note: the ``destination\_data\_column`` and ``destination\_key\_column`` arguments should be omitted when using a volume as destination. Volumes behave like key/value stores so there are no column names to configure.

```
SELECT aidb.create_volume_preparer(
    name => chunking_example_volumes,
    operation => ChunkText,
    source_volume_name => text_files_volume,
    destination_table => chunk_output_volume,
    options => {"desired_length": 1000}::JSONB
);

SELECT aidb.bulk_data_preparation(chunking_example_volumes);
```

Inspect the results

List the results from within AIDB:

```
SELECT * FROM aidb.list_volume_content(chunk_output_volume);
```

__OUTPUT__	file_name	size	last_modified
	phil.txt.part.17	4	2025-05-02 15:56:50.161647402 UTC
	bilge.txt.part.24	9	2025-05-02 15:56:49.885069607 UTC
	mark.txt.part.5	4	2025-05-02 15:56:50.025494534 UTC
	phil.txt.part.28	4	2025-05-02 15:56:50.179620037 UTC
	phil.txt.part.10	97	2025-05-02 15:56:50.150750239 UTC
	mark.txt.part.2	3	2025-05-02 15:56:50.020690984 UTC
	TianQiu.txt.part.9	13	2025-05-02 15:56:49.809999366 UTC
	bilge.txt.part.23	5	2025-05-02 15:56:49.883231553 UTC
	phil.txt.part.26	8	2025-05-02 15:56:50.176445170 UTC
	TianQiu.txt.part.0	8	2025-05-02 15:56:49.792321435 UTC
	phil.txt.part.19	8	2025-05-02 15:56:50.165073307 UTC
	bilge.txt.part.15	12	2025-05-02 15:56:49.869647891 UTC
...			
	julien.txt.part.2	6	2025-05-02 15:56:49.909471517 UTC
	julien.txt.part.5	4	2025-05-02 15:56:49.914235651 UTC

(269 rows)

We can see that each chunk is stored as a `part` with a sequential ID.

Capabilities

Pipeline lifecycle

This high-level overview describes the lifecycle of a pipeline in AI Accelerator Pipelines.

Optionally, create a storage location

This step is optional and needed only for accessing data external storage.

Data for processing can be stored in the database in a table or in an external storage location. If you want to use an external storage location to access data, you must create a storage location. This storage location can be an S3 bucket or a local file system.

The storage locations can be used by AI Accelerator to create a volume. This volume can then be used by a knowledge base to access its data.

Create a preparer (optional)

This step is optional and needed only for preprocessing source AI data.

Create a preparer with the Pipelines system. A preparer is a function that retrieves data from a table or volume and returns it in a format that can be used by the model.

With any configuration, a preparer requires:

- A name
- The data preparation operation
- The name of the destination table where the output data will be stored
- The name of the column in the destination table where the output data will be stored

If the preparer is for a table, it also needs:

- The name of the source table
- The name of the column in the source table that contains the data

But if the preparer is for a volume, it needs:

- The name of the volume

When a preparer is created, by default it assumes column identifiers of “id” for the source and destination, but this can be customized.

Create a model

Create a *Pipelines models reference* with AI Accelerator Pipelines. This model can be any AI model suitable for your tasks.

Create a knowledge base

Create a knowledge base with AI Accelerator Pipelines. A knowledge base is a function that retrieves data from a table or volume and returns it in a format that the model can use.

By default, a knowledge base only needs:

- A name
- The name of a model to use

If the knowledge base is for a table, it also needs:

- The name of the source table
- The name of the column in the source table that contains the data
- The data type of the column

If the knowledge base is for a volume, it needs:

- The name of the volume
- The name of the column in the volume that contains the data

When you create a knowledge base, by default a vector table is created to store the embeddings of the data that's retrieved. This table has a column to store the embeddings and a column to store the key of the data.

When you create the knowledge base, you can specify the name of the vector table and the name of the vector column and the key column. This ability is useful if you're migrating to aidb and want to use an existing vector table.

Create embeddings

Embedding sees the data being retrieved from the source table or volume and encoded into a vector datatype. That vector data is then stored in the vector table.

See [Pipeline auto-processing](#) to understand how embedding computation can be configured and run.

Query data

You can query the embedded data using the knowledge base. The knowledge base can return the key to the data or the data itself, depending on the query. You can query the data using a text query or an image query, depending on the type of data that's being retrieved.

Next steps

While auto-processing is enabled, the embeddings are always up to date, and applications can use the knowledge base to query the data as needed.

Cleanup

If the embeddings are no longer required, you can delete the knowledge base, drop the vector table, and delete the model.

Pipeline auto-processing

Overview

Pipeline auto-processing is designed to keep source data and pipeline output in sync. Without this capability, you would have to manually trigger processing or provide external scripts, schedulers, or triggers. Auto-processing includes these features:

- **Full sync:** Insert/delete/update are all handled automatically. No lost updates, missing data, or stale records.
- **Change detection:** Only new or changed records are processed. No unnecessary reprocessing of known records.
- **Batch processing:** Records are grouped into batches to be processed concurrently. Reduces overhead and achieves optimal performance, for example, with GPU-based AI model inference tasks.
- **Background processing:** When background processing is enabled, the pipeline runs in a background worker process so that it doesn't block or delay other DB operations. Ideal for processing huge datasets.
- **Live processing for Postgres tables:** When the data source is a Postgres table, live trigger-based auto-processing can be enabled so that pipeline results are always up to date.
- **Quick turnaround:** Once a batch has finished processing, the results are immediately available. No full listing of the source data is needed to start processing, which is important for large external volumes where a full listing can take a long time.

Example for knowledge base pipeline

Suppose a knowledge base is created for a Postgres table containing products with product descriptions. The user configures background auto-processing to always keep embeddings in sync without blocking or delaying any operations on the products table.

The pipeline processes any preexisting product records in the background. The user can query the statistics table to see the progress.

The background process runs when new data is inserted or existing data is modified or deleted.

Queries on the knowledge base (that is, retrieval operations) always return accurate results after a small background processing delay.

Supported pipelines and modes

Knowledge base pipeline

Source type	Destination type	Live	Background	Disabled (manual)
Table	Table	OK	OK	OK
Volume	Table	NO	OK	OK

Outputting to volumes isn't supported on knowledge base pipelines. A database with vector capabilities is needed.

Preparer pipeline

Source type	Destination type	Live	Background	Disabled (manual)
Table	Table	OK	NO	OK
Table	Volume	NO	NO	OK
Volume	Table	NO	NO	OK
Volume	Volume	NO	NO	OK

The preparer pipeline doesn't yet support batch processing and background auto-processing.

Auto-processing modes

The following auto-processing modes are available to suit different requirements and use cases.

Live

AIDB sets up Postgres triggers on the source table to immediately process any changes. Processing happens in the trigger function. This means it happens in the same transaction that modifies the data so that results are up to date.

Pros & cons

- Transactional guarantee/immediate results. Pipeline results are always up to date with the source data.
- Blocks/delays operations on the source data. Modifying transactions on the source data are delayed until processing is complete.

Background

AIDB starts a Postgres background worker for each pipeline that has background auto-processing configured. Processing happens asynchronously based on a configurable `background_sync_interval`. See [Change detection](#) for details on how the pipelines are processed.

Note

Make sure Postgres allows running enough background workers for the number of pipelines where you want to use this processing mode. Use the Postgres setting `max_worker_processes` to control this value.

Pros & cons

- Asynchronous execution means queries on the source don't have to be delayed while the changes are processed.
- Results are delayed and might become backlogged.
- Ideal for huge datasets. Processing occurs continuously in the background and isn't tied to any user session or SQL function call.

Disabled

Auto-processing is disabled. You can manually call `aidb.bulk_embedding()` to process the pipelines.

Note

On table knowledge bases, change detection is also disabled, since it requires active triggers on the source table. This means manual processing (via `aidb.bulk_embedding()`) has to process all the records in the source.

Observability

You can see detailed status and progress output for all auto-processing modes.

A good place to get an overview is the statistics table. Look up the view `aidb.knowledge_base_stats`, or use its short alias `aidb.kbstat`. The view shows all configured knowledge base pipelines, which processing mode is set, and statistics about the processed records:

```
SELECT * from aidb.kbstat;
__OUTPUT__
  knowledge base      | auto processing | table: unprocessed rows | volume: scans_
  completed | count(source records) | count(embeddings)
-----+-----+-----+-----+
kb_table_text_bg      | Background     | 15 | 0 |
kb_table_text_manual  | Disabled       | 15 | 0 |
kb_table_image_manual | Disabled       | 3  | 0 |
kb_table_text_live    | Live           | 15 | 0 |
kb_table_image_bg     | Background     | 3  | 0 |
```

(continues on next page)

(continued from previous page)

kb_volume_text_bg	Background			
→ 6	7		7	└
kb_volume_text_manual	Disabled			└
→ 0	0		0	
kb_volume_image_bg	Background			└
→ 4	177		6	
kb_volume_image_manual	Disabled			└
→ 1	177		6	
(9 rows)				

The *Change detection* mechanism is central to how auto-processing works. It's different for volume and table sources. For this reason, the stats table has different columns for these two source types.

- `table: unprocessed rows` : How many unique rows are listed in the backlog of change events. * If auto-processing is disabled, no (new) change events are captured.
- `volume: scans completed` : How many full listings of the source have been completed so far.
- `count(source records)` : How many records exist in the source for this pipeline. * For table sources, this number is always accurate. * For volume sources, this number is updated only after a full scan has completed.
- `count(embeddings)` : How many embeddings exist in the vector destination table for this pipeline.

Configuration

You can configure auto-processing at creation time:

- With `aidb.create_table_knowledge_base`
- With `aidb.create_volume_knowledge_base`

You can also configure it for existing pipelines:

- With `aidb.set_auto_knowledge_base`

Batch processing

In background and disabled modes, (auto) processing happens in batches of configurable size. The pipeline processes all source records in batches. All records in each batch are processed in parallel wherever possible. This means pipeline steps like data retrieval, embeddings computation, and storing embeddings run as parallel operations. For example, when using a table as a data source, a batch of input records is retrieved with a single query. With a volume source, concurrent requests are used to retrieve a batch of records.

Pipelines knowledge base performance tuning explains how you can tune the batch size for optimal throughput.

Change detection

AIDB auto-processing is designed around change detection mechanisms for table and volume data sources. Change detection allows it to process data only when necessary.

Table sources

When background auto-processing is enabled, Postgres triggers are set up on the source table to detect changes. These triggers are very lightweight. They only record change events and insert them into a “change events” table. No actual processing happens in the trigger function.

The background worker then processes these events asynchronously.

Volume sources

This source type provides a `last_modified` timestamp for each source record. The system keeps track of those timestamps in a “state” table. In each pipeline execution, the system lists the contents of the volume and compares it to the timestamps to see whether any records have changed or were added.

The system detects deleted objects after a full listing is complete. Only then can it be certain that a previously processed record is no longer present in the source.

Note

Unfortunately, object stores (and other external storage locations supported by our volumes) have limited query capabilities. This means: Change detection for volumes is based on polling, that is, repeated listing. This might be an expensive operation when using cloud object stores like AWS S3. You can use a long `background_sync_interval` (like one per day) on pipelines with volume sources to control this cost.

GPU-accelerated vector index build

Overview

Building a vector index is a compute-intensive process that can run for hours or days on very large tables. EDB Postgres AI – AI Accelerator (aidb) can use NVIDIA GPUs to accelerate vector index creation for the `VectorChordq` index type.

Use GPU acceleration when creating indexes on large vector columns to significantly reduce build time.

Requirements

Before using GPU acceleration, ensure the environment meets these requirements:

- Supported OS: Ubuntu 22.04/24.04, Debian 12, RHEL 9.
- CPU architecture: x86_64.
- Supported NVIDIA GPUs: architectures 8.9 and 9.0 (for example, L40, L4, H100, H200, GH200).
- NVIDIA GPU driver installed and functional.
- NVIDIA CUDA runtime 12.9 or newer installed.
- The VectorChord extension available on the system (for the `vectorchordq` index type). See the [Vector Engine documentation](#) for install and usage details.

See also: general platform support in [API Compatibility](#).

Enable GPU acceleration

GPU acceleration is disabled by default. Enable it per session or at the server level.

- Per session:

```
SET aidb.gpu_acceleration = enable;
```

- Persist in the server configuration and reload:

```
ALTER SYSTEM SET aidb.gpu_acceleration = enable;
SELECT pg_reload_conf();
```

To disable, set the value to `disable` and reload the configuration.

Quick start example

The following example creates a table with a vector column, inserts sample data, and builds a GPU-accelerated VectorChord index.

1) Create a test table:

```
CREATE TABLE test_10k_vecs (
  id          bigserial PRIMARY KEY,
  embedding   vector(2000)
);
```

2) Insert sample vectors:

```
INSERT INTO test_10k_vecs (embedding)
SELECT arr.embedding
FROM generate_series(1, 10000) AS g(i)
CROSS JOIN LATERAL (
  SELECT array_agg(((g.i - 1) * 3 + gs.j)::real)
  FROM generate_series(1, 2000) AS gs(j)
) AS arr(embedding);
```

3) Build the GPU-accelerated index:

```
SELECT aidb.create_vector_index_on_gpu(
  table_name      => public.test_10k_vecs,
  column_name     => embedding,
  cluster_count   => 1000
);
```

When the function completes, the table has a VectorChord `vchordq` index ready for use.

Function reference

The `aidb.create_vector_index_on_gpu()` function runs the entire VectorChord index build on the GPU. It is analogous to `CREATE INDEX` for VectorChord.

Signature example with all parameters:

```
SELECT aidb.create_vector_index_on_gpu(
  table_name      => public.test_10k_vecs,
  column_name     => embedding,
  cluster_count   => 1000,
  sampling_factor  => 256,
  kmeans_iterations => 10,
  kmeans_nredo    => 1,
  distance_operator => ip,
  batch_size      => 1000000
);
```

Parameters:

- `table_name`: Fully qualified table name (`schema.table`).
- `column_name`: Name of the vector column to index.
- `cluster_count`: Number of clusters to use for the index (VectorChord “lists”).

- `sampling_factor`: Optional. Number of sample vectors for `k-means`. Default: 256.
- `kmeans_iterations`: Optional. Number of `k-means` iterations. Default: 10.
- `kmeans_nredo`: Optional. Number of `k-means` restarts. Default: 1.
- `distance_operator`: Optional. Distance metric. One of `ip` (inner product), `l2` (Euclidean), or `cos` (cosine). Default: `ip`.
- `batch_size`: Optional. Controls processing batch size. Larger batches increase memory use but can improve throughput.

Verification

After the function returns, verify that the index exists and is usable.

- List indexes on the table:

```
SELECT indexname, indexdef
FROM pg_indexes
WHERE schemaname = public AND tablename = test_10k_vecs;
```

- Run an example query with your intended distance operator and use `EXPLAIN` to confirm index usage.

Verify index usage with EXPLAIN

Replace `:query_vec` with a query vector literal or parameter of the same dimension as your column. Choose the operator that matches the `distance_operator` you used for index creation.

- L2 distance (`l2`):

```
EXPLAIN
SELECT id
FROM test_10k_vecs
ORDER BY embedding <-> :query_vec
LIMIT 10;
```

- Inner product (`ip`):

```
EXPLAIN
SELECT id
FROM test_10k_vecs
ORDER BY embedding <#> :query_vec
LIMIT 10;
```

- Cosine distance (`cos`):

```
EXPLAIN
SELECT id
FROM test_10k_vecs
ORDER BY embedding <=> :query_vec
LIMIT 10;
```

The plan should indicate an index-assisted path (for example, Index Scan) rather than a full sequential scan. If needed for testing, temporarily set `SET enable_seqscan = off;` to encourage index usage while validating.

For details on vector distance operators `<->` (L2), `<#>` (inner product), and `<=>` (cosine), see the [Vector Engine](#) documentation.

Tuning and guidance

- Cluster count: Choose a value appropriate for your data size and query patterns, following VectorChord guidance for the number of lists.
- Batch size: Increase for faster builds when sufficient GPU memory is available; reduce if you encounter out of memory conditions.
- Distance operator: Match to how you plan to query the data (`ip`, `l2`, or `cos`).

Troubleshooting

- GPU acceleration not enabled: Ensure `aidb.gpu_acceleration = 'enable'` and reload configuration if set via `ALTER SYSTEM`.
- No compatible GPU detected: Verify NVIDIA drivers and CUDA runtime installation.
- Missing VectorChord components: Ensure the VectorChord extension and its index type are installed on the node.
- Invalid arguments: Confirm the vector column exists, has a fixed dimension, and the table name is fully qualified.

Notes and limitations

- GPU acceleration currently applies to VectorChord `vhordq` index builds.
- Supported hardware and OS are limited to the platforms listed in Requirements.
- The function runs synchronously and returns when the index build completes or fails.

Pipelines models

Pipelines has a model registry that manages configured instances of models. Any Pipelines functions that use models, such as embedding and retrieving, must reference a created model.

- How to *Using models in AI Accelerator Pipelines* in Pipelines.
- Discover the *Pipelines model primitives* that can be used to interact with models.
- See the *Supported models* that come with Pipelines.
- Using models with OpenAI API-compatible services and Nvidia NIM with Pipelines.

Next steps

Once you are familiar with models, you can learn how to use those models with *Knowledge Bases*.

Using models in AI Accelerator Pipelines

Pipelines has a model registry that manages configured instances of models. Any Pipelines functions that use models, such as embedding and retrieving, must reference a model in this registry.

Discover the preloaded models

Pipelines comes with a set of preconfigured models that you can use out of the box.

To find them, run the following query:

```
SELECT * FROM aidb.models;
```

This query returns a list of all the models that are currently created in the system. If you haven't created any models, you'll see the default models that come with Pipelines.

name	provider	options
bert	bert_local	{"config={}"}
clip	clip_local	{"config={}"}
t5	t5_local	{"config={}"}
dummy	dummy	{"config={}"}

The `bert`, `clip`, and `t5` models are all preconfigured and ready to use. The `dummy` model is a placeholder model that can be used for testing purposes.

If you use a proxy, ensure that it's *configured on the Postgres server*.

You can create your own models. To do this, you can use the `aidb.create_model` function. This example shows how to create a model:

```
SELECT aidb.create_model(my_model, bert_local);
```

This command creates a model named `my_model` that uses the default `bert_local` model provider. Because this example doesn't use any additional parameters, `my_model` is the same as the `bert_local` model that came preloaded. To configure the model when you create it, see *Creating a model with a configuration*.

Discovering the model providers

You can find the model providers that are available by running the following query:

```
SELECT * FROM aidb.model_providers;
```

server_name	server_description	server_options
t5_local	A simple language model, ideal for translation , summarization, and question answering. Runs locally.	
embeddings	For any model that implements the OpenAI embeddings API.	
completions	For any model that implements the OpenAI completions API.	
openai_embeddings	For any embeddings model on the OpenAI platform	
openai_completions	For any completions model on the OpenAI platform.	
nim_completions	For any model that implements the OpenAI completions API on NVIDIA NIM.	
nim_embeddings	For any model that implements the OpenAI embeddings API on NVIDIA NIM.	
nim_clip	Vision/text model, ideal for encoding text and images , runs on NVIDIA NIM.	
nim_reranking	Reranking model, runs on NVIDIA NIM.	
bert_local	Simple language model, ideal for encoding text , runs locally.	
clip_local	A vision/text model, ideal for encoding text and images , runs	

(continues on next page)

(continued from previous page)

```

→ locally.
dummy          | Provides fake data, ideal for testing.
→
(12 rows)

```

This query returns a list of all the model providers that are currently available in the system. You can find out more about these providers and their capabilities in [Supported models](#).

Creating a model with a configuration

You can pass options to the model when creating it. For example, you can specify the model configuration:

```

SELECT aidb.create_model(my_model,
                        bert_local,
                        {"model": "sentence-transformers/paraphrase-multilingual-
→MiniLM-L12-v2",
                        "revision": "main"}::JSONB);

```

This command creates a model named `my_model` that uses the `bert_local` model provider and the `sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2` model from HuggingFace. The `revision` option specifies the version of the model to use. The options are passed as a JSONB object, with a single quoted string that's then cast to JSONB. In the string are the key-value pairs that define the model configuration in a single JSON object.

Creating a model with configuration and credentials

Using a configuration and credentials is where the other [Supported models](#) come in. You can create a different model by specifying the model name in the configuration. The `openai-completions` and `openai-embeddings` models are both models that you can create to make use of OpenAI's completions and embeddings APIs. Similarly, the `nim-completions` and `nim-embeddings` models are models that you can create to make use of NVIDIA's completions and embeddings APIs.

You need to provide more information to the `aidb.create_model` function when creating a model this way. Completions has a number of options, including selecting the model it will use on OpenAI. Both completions and embeddings require API credentials. This example shows how to create the OpenAI completions model:

```

SELECT aidb.create_model(
  my_openai_model,
  openai_completions,
  {"model": "gpt-4o"}::JSONB,
  {"api_key": "sk-abc123xyz456def789ghi012jkl345mn"}::JSONB
);

```

Replace the `api_key` value with your own OpenAI API key. (For more information about credential handling, see [Supported models](#).) Then you can use the `my_openai_model` model in your Pipelines functions and, in this example, leverage the GPT-4o model from OpenAI.

Creating the OpenAI embeddings model is similar to creating the completions model:

```

SELECT aidb.create_model(
  my_openai_embeddings,
  openai_embeddings,
  {"model": "text-embedding-3-small"}::JSONB,

```

(continues on next page)

(continued from previous page)

```
{ "api_key": "sk-abc123xyz456def789ghi012jkl345mn" }::JSONB
);
```

This command creates the `text-embedding-3-small` model with the name `my_openai_embeddings`. You can then use this model in your Pipelines functions to generate embeddings for text data.

Where models do use credentials, the credentials set in the first use of `create_model` for the model provider are used with all subsequent uses of that model. You can't use different credentials for different models that use the same provider.

If you need to change the credentials, you can use the `replace_credentials` option when creating a new model. That option updates the credentials for the model provider.

```
SELECT aidb.create_model(
  my_new_openai_model,
  openai_completions,
  { "model": "gpt-4o" }::JSONB,
  { "api_key": "sk-ngl234meh789ski1111bid011i" }::JSONB,
  replace_credentials => true
);
```

If you only needed to change the credentials, you can delete the new model using `SELECT aidb.delete_model('my_new_openai_model')`.

Using models with OpenAI-compatible APIs

The completions and embeddings models work with any OpenAI-compatible API. This compatibility allows you to connect and use an even wider range of models. Pass the appropriate API endpoint to the `url` option in the `aidb.create_model` function's options.

For more information about the models, see [Completions](#) and [Embeddings and vector representations](#).

Pipelines model primitives

For most use cases, we recommend that you use the `aidb Knowledge Bases` to interact with models. They can manage creating embeddings and retrieving matching data for many applications.

However, if you need to interact with models directly, you can use the following primitives. The `encode` functions generate embeddings for text and images, and the `decode` functions generate text from embeddings.

Generate text embeddings

Call `aidb.encode_text` to generate a vector representation of a given text input:

```
select aidb.encode_text(t5, This is text I need vectorized.);
```

For batch inferencing, call `aidb.encode_text_batch`:

```
select aidb.encode_text_batch(my_bert_model, ARRAY[
  Dwayne "The Rock" Johnson,
  Hatsune Miku,
  Jim Lehrer,
  Dikembe Mutombo
]);
```

Generate image embeddings

Call `aidb.encode_image` to generate a vector representation of a given image input:

```
-- BYTEA type casting included to demonstrate what data type is used
select aidb.encode_image(clip, image1 AS BYTEA);
```

For batch inferencing, call `aidb.encode_image_batch`:

```
-- BYTEA type casting included to demonstrate what data type is used
select aidb.encode_image_batch(clip, ARRAY[
  image_of_the_rock_cooking AS BYTEA,
  image_of_miku_singing AS BYTEA,
  image_of_jim_reporting_news AS BYTEA,
  image_mutombo_blocking_shots AS BYTEA
]);
```

Generate prompt responses

Call `aidb.decode_text` to generate a prompt response from a given text input:

```
SELECT * FROM aidb.decode_text(t5, translate to german: hello world);
__OUTPUT__
decode_text
-----
Hallo Welt
(1 row)
```

For batch inferencing, call `aidb.decode_text_batch`:

```
SELECT * FROM aidb.decode_text_batch(t5, ARRAY[
  translate to german: i am an example,
  summarize: The missile knows where it is at all times. It knows this because it_
↳ knows where it isnt. By subtracting where it is from where it isnt, or where it_
↳ isnt from where it is (whichever is greater), it obtains a difference, or deviation.
↳ The guidance subsystem uses deviations to generate corrective commands to drive_
↳ the missile from a position where it is to a position where it isnt, and arriving_
↳ at a position where it wasnt, it now is.
]);
__OUTPUT__
↳ decode_text_batch
-----
↳
↳
ich bin ein Beispiel
the guidance subsystem uses deviations to generate corrective commands . it is_
↳ driving the missile from a position where it is, or where it isnt, and arriving at_
↳ a position where it wasnt .
(2 rows)
```

Rerank text

Call `aidb.rerank_text` to get text reranking logits:

```
SELECT * FROM aidb.rerank_text(my_nim_reranker, how can I open a door?, {'Ask for help,
→ Push the handle, Lie down and wait, Shout at it'}::text[]) ORDER BY logit_score_
→DESC;
__OUTPUT__
      text      | logit_score | id
-----+-----+---
Push the handle | -3.697265625 | 1
Ask for help    | -6.2578125   | 0
Shout at it     | -7.39453125  | 3
Lie down and wait | -11.375      | 2
(4 rows)
```

Pipelines supported models

This section provides details of the supported models in EDB Postgres AI - AI Accelerator Pipelines and their capabilities.

- *Support matrix* for embedded models.
- *T5* .
- *Embeddings and vector representations* , including openai-embeddings and nim-embeddings .
- *Completions* , including openai-completions and nim-completions .
- *Gemini*
- *BERT* .
- *CLIP* .
- *NIM_CLIP* .
- *Llama* .
- *Reranking (NIM)* .
- *NIM OCR* .

Support matrix

Model support matrix

Overview of the *embedded* models that are supported by the AI Factory for various operations. For more details about each model, see the specific page for that model provider.

Provider	Model name	Operations	Configuration
<i>bert_local</i>	<i>bert_all-MiniLM-L12-v1</i>	<i>TextEncoder</i>	<i>{"model": "sentence-transformers/all-MiniLM-L12-v1", "revision": "main"}</i>
<i>bert_local</i>	<i>bert_all-MiniLM-L6-v1</i>	<i>TextEncoder</i>	<i>{"model": "sentence-transformers/all-MiniLM-L6-v1", "revision": "main"}</i>
<i>bert_local</i>	<i>bert_all-MiniLM-L6-v2</i>	<i>TextEncoder</i>	<i>{"model": "sentence-transformers/all-MiniLM-L6-v2", "revision": "main"}</i>
<i>bert_local</i>	<i>bert_all-distilroberta-v1</i>	<i>TextEncoder</i>	<i>{"model": "sentence-transformers/all-distilroberta-v1", "revision": "main"}</i>
<i>bert_local</i>	<i>bert_msmarco-bert-base-dot-v5</i>	<i>TextEncoder</i>	<i>{"model": "sentence-transformers/msmarco-bert-base-dot-v5", "revision": "main"}</i>
<i>bert_local</i>	<i>bert_multi-qa-MiniLM-L6-cos-v1</i>	<i>TextEncoder</i>	<i>{"model": "sentence-transformers/multi-qa-MiniLM-L6-cos-v1", "revision": "main"}</i>
<i>bert_local</i>	<i>bert_multi-qa-MiniLM-L6-dot-v1</i>	<i>TextEncoder</i>	<i>{"model": "sentence-transformers/multi-qa-MiniLM-L6-dot-v1", "revision": "main"}</i>
<i>bert_local</i>	<i>bert_paraphrase-TinyBERT-L6</i>	<i>TextEncoder</i>	<i>{"model": "sentence-transformers/paraphrase-TinyBERT-L6", "revision": "main"}</i>
<i>bert_local</i>	<i>bert_paraphrase-multilingual-v1</i>	<i>TextEncoder</i>	<i>{"model": "sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2", "revision": "main"}</i>
<i>bert_local</i>	<i>bert_paraphrase-multilingual-n</i>	<i>TextEncoder</i>	<i>{"model": "sentence-transformers/paraphrase-multilingual-mpnet-base-v2", "revision": "main"}</i>
<i>clip_local</i>	<i>openai/clip-vit-base-patch32</i>	<i>TextEncoder, ImageEncoder</i>	<i>{}</i>
<i>llama_instruct_local</i>	<i>Hugging-FaceTB/SmollM2-1.7B-Instruct</i>	<i>TextDecoder</i>	<i>{"model": "Hugging-FaceTB/SmollM2-1.7B-Instruct", "revision": "main"}</i>
<i>llama_instruct_local</i>	<i>Hugging-FaceTB/SmollM2-1.7B-Instruct</i>	<i>TextDecoder</i>	<i>{"model": "Hugging-FaceTB/SmollM2-1.7B-Instruct", "revision": "main"}</i>
<i>llama_instruct_local</i>	<i>Hugging-FaceTB/SmollM2-1.7B-Instruct</i>	<i>TextDecoder</i>	<i>{"model": "Hugging-FaceTB/SmollM2-1.7B-Instruct", "revision": "main"}</i>
<i>llama_instruct_local</i>	<i>Hugging-FaceTB/SmollM2-1.7B-Instruct</i>	<i>TextDecoder</i>	<i>{"model": "Hugging-FaceTB/SmollM2-1.7B-Instruct", "revision": "main"}</i>
<i>llama_instruct_local</i>	<i>TinyLlama/TinyLlama-1.1B-Chat-v1.0</i>	<i>TextDecoder</i>	<i>{"model": "TinyLlama/TinyLlama-1.1B-Chat-v1.0", "revision": "main"}</i>
<i>t5_local</i>	<i>t5-base</i>	<i>TextEncoder, TextDecoder</i>	<i>{"model": "t5-base", "revision": "main"}</i>
<i>t5_local</i>	<i>t5-large</i>	<i>TextEncoder, TextDecoder</i>	<i>{"model": "t5-large", "revision": "main"}</i>
<i>t5_local</i>	<i>t5-small</i>	<i>TextEncoder, TextDecoder</i>	<i>{"model": "t5-small", "revision": "main"}</i>

T5

Model name: `t5_local`

About T5

T5 is a text-to-text transformer model that converts input text to output text. It's based on the Transformer architecture where an encoder processes the input and a decoder then generates the output. It's pre-trained on a large corpus of text data and is capable of performing a wide range of natural language processing tasks, such as chatbots, translation, summarization, and question answering.

Read more about [T5 on Wikipedia](#)).

Supported aidb operations

- `encode_text`
- `decode_text`
- `encode_text_batch`
- `decode_text_batch`

Supported models

- `t5-small` (default)

See more in the *Support matrix* .

Creating the default model

```
SELECT aidb.create_model(my_t5_model, t5_local);
```

Creating a specific model

```
SELECT aidb.create_model(
  another_t5_model,
  t5_local,
  [{"model": "t5-large", "revision": "main"}]::JSONB
)
```

Model configuration settings

The following configuration settings are available for T5 models:

- `model` — The T5 model to use. The default is `t5-small` .
- `revision` — The revision of the model to use. The default is `refs/pr/15` .
- `temperature` — The temperature to use for sampling. The default is `0.0` .
- `seed` — The random seed to use for sampling. The default is `1599222198345926291` .
- `max_tokens` — The maximum number of tokens to generate. The default is `16384` .
- `repeat_penalty` — The repetition penalty to use. The default is `1.1` .
- `repeat_last_n` — The number of tokens to consider for the repetition penalty. The default is `64` .

Model credentials

No credentials are required for the T5 model.

BERT

Model name: `bert_local`

About BERT

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model that's used for natural language processing tasks such as text classification, text generation, and text completion. It's pre-trained on a large corpus of text data and is capable of performing a wide range of natural language processing tasks.

Read more about [BERT on Wikipedia](#)) and at [Sentence Transformers](#) on the Hugging Face website.

Supported aidb operations

- `encode_text`
- `encode_text_batch`

Supported models

- `sentence-transformers/all-MiniLM-L6-v2` (default)

See more in the *Support matrix* .

Creating the default model

```
SELECT aidb.create_model(my_bert_model, bert_local);
```

Creating a specific model

You can specify a model and revision in the options JSONB object. This example creates a `sentence-transformers/all-distilroberta-v1` model with the name `another_bert_model` :

```
select aidb.create_model(
  another_bert_model,
  bert_local,
  {"model": "sentence-transformers/all-distilroberta-v1", "revision": "main"}::JSONB
)
```

Model configuration settings

The following configuration settings are available for BERT models:

- `model` — The BERT model to use. The default is `sentence-transformers/all-MiniLM-L6-v2` .
- `revision` — The revision of the model to use. The default is `refs/pr/64` . This entry is a reference to the model revision in the Hugging Face repository. It's used to specify the model version to use, in this case [this branch](#) .

Model credentials

No credentials are required for the BERT models, as they run locally.

CLIP

Model name: `clip_local`

About CLIP

CLIP (Contrastive Language-Image Pre-training) is a model that learns visual concepts from natural language supervision. It's a zero-shot learning model that can be used for a wide range of vision and language tasks.

Read more about *CLIP on OpenAI's website* .

Supported aidb operations

- `encode_text`
- `encode_text_batch`
- `encode_image`
- `encode_image_batch`

Supported models

- `openai/clip-vit-base-patch32` (default)

See more in the *Support matrix* .

Creating the default model

```
SELECT aidb.create_model(my_clip_model, clip_local);
```

There's only one model, the default `openai/clip-vit-base-patch32` , so you don't need to specify the model in the configuration. No credentials are required for the CLIP model.

Creating a specific model

There are no other model configurations available for the CLIP model.

Model configuration settings

The following configuration settings are available for CLIP models:

- `model` — The CLIP model to use. The default is `openai/clip-vit-base-patch32` and is the only model available.
- `revision` — The revision of the model to use. The default is `refs/pr/15` . This entry is a reference to the model revision in the Hugging Face repository. It's used to specify the model version to use, in this case *this branch* .
- `image_size` — The size of the image to use. The default is `224` .

Model credentials

No credentials are required for the CLIP model.

Llama

Model name: llama_instruct_local

About Llama

LLaMA is a decoder-only transformer model designed for text generation tasks. It uses the standard Transformer architecture without an encoder, processing input tokens autoregressively to predict the next token in sequence. Pre-trained on a large-scale corpus of publicly available text, LLaMA is capable of handling various natural language tasks, including chat, code generation, summarization, and question answering.

Read more about [Llama on Wikipedia](#)).

Supported aidb operations

- decode_text
- decode_text_batch

Supported models

See more in the *Support matrix* .

Creating the default model

```
SELECT aidb.create_model(my_llama_model, llama_instruct_local);
```

Creating a specific model

```
SELECT aidb.create_model(
  another_llama_model,
  llama_instruct_local,
  {"model": "HuggingFaceTB/SmolLM2-135M-Instruct", "revision": "main"}::JSONB
)
```

Running the model

```
SELECT aidb.decode_text(llama_instruct, Why is the sky blue?);
```

Model configuration settings

The following configuration settings are available for Llama models:

- model — The Llama model to use. The default is TinyLlama/TinyLlama-1.1B-Chat-v1.0 .
- revision — The revision of the model to use. The default is main .
- system_prompt — Optional. Foundational instructions to guide general LLM responses.
- use_flash_attention — Indicate if the model uses flash attention. The default is false .

- `seed` — The random seed to use for sampling. The default is 1599222198345926291 .
- `temperature` — The temperature to use for sampling. The default is 0.2 .
- `sample_len` — The maximum number of tokens to generate. The default is 64 .
- `repeat_last_n` — The number of tokens to consider for the repetition penalty. The default is 64 .
- `repeat_penalty` — The repetition penalty to use. The default is 1.1 .
- `top_p` — Cumulative probability threshold for filtering the token distribution. The default is 0.9 .
- `use_kv_cache` — Enables reuse of attention key/value pairs during generation for faster decoding. The default is `true` .

Model credentials

No credentials are required for local Llama models.

Gemini

Model name: `gemini`

About Gemini

Google Gemini is a family of multimodal large language models developed by Google. The models are accessible via the Gemini API.

Supported aidb operations

- `decode_text`
- `decode_text_batch`

Supported models

Any text generation model that the provider supports. See a list of supported Gemini models [here](#) .

Creating a Gemini model

You can create any supported Gemini model using the `aidb.create_model` function.

This example creates a Gemini 2.5 Flash model with the name `my_gemini_model` :

```
SELECT aidb.create_model(
  my_gemini_model,
  gemini,
  {"model": "gemini-2.5-flash"}::JSONB,
  {"api_key": "AIzaSyBXnKr4yJG2lMhyvza81q3-WFuP7Tk3r8z"}::JSONB
);
```

Running the model

```
SELECT aidb.decode_text(my_gemini_model, Why is the sky blue?);
```

Model configuration settings

The following configuration settings are available for OpenAI models:

- `model` — The model to use.
- `url` — The URL of the model to use. This setting is optional and can be used to specify a custom model URL.
- `max_concurrent_requests` — The maximum number of concurrent requests to make to the OpenAI model. The default is 25.
- `thinking_budget` — The number of thinking tokens used to generate responses. This setting is optional. See more [here](#).

Model credentials

The following credentials may be required by the service providing these models.

- `api_key` — The API key to use for Bearer Token authentication. The `api_key` is sent in a header field as `Authorization: Bearer <api_key>`.

Completions

Model name: `completions`

Model aliases:

- `openai_completions`
- `nim_completions`

About completions

Completions enables the use of any OpenAI API-compatible text-generation model. It's suitable for chat/text transforms, text completion, and other text generation tasks.

Based on the name of the model, the model provider sets defaults accordingly:

- When invoked as `completions` or `openai_completions`, the model provider defaults to using the OpenAI API.
- When invoked as `nim_completions`, the model provider defaults to using the NVIDIA NIM API.

Supported aidb operations

- `decode_text`
- `decode_text_batch`

Supported models

- Any text generation model that's supported by the provider.

Supported OpenAI models

See a list of supported OpenAI models [here](#).

Supported NIM models

- `ibm/granite-guardian-3.0-8b`
- `ibm/granite-3.0-8b-instruct`
- `ibm/granite-3.0-3b-a800m-instruct`
- `meta/llama-3.3-70b-instruct`
- `meta/llama-3.2-3b-instruct`
- `meta/llama-3.2-1b-instruct`
- `meta/llama-3.1-405b-instruct`
- `meta/llama-3.1-70b-instruct`
- `meta/llama-3.1-8b-instruct`
- `meta/llama3-70b-instruct`
- `meta/llama3-8b-instruct`
- `nvidia/llama-3.1-nemotron-70b-instruct`
- `nvidia/llama-3.1-nemotron-51b-instruct`
- `nvidia/nemotron-mini-4b-instruct`
- `nvidia/nemotron-4-340b-instruct`
- `google/shieldgemma-9b`
- `google/gemma-7b`
- `google/codegemma-7b`

Creating the default model

There's no default model for completions. You can create any supported model using the `aidb.create_model` function.

Creating an OpenAI model

You can create any supported OpenAI model using the `aidb.create_model` function.

This example creates a GPT-4o model with the name `my_openai_model`:

```
SELECT aidb.create_model(
  my_openai_model,
  openai_completions,
  {"model": "gpt-4o"}::JSONB,
  {"api_key": "sk-abc123xyz456def789ghi012jkl345mn"}::JSONB
);
```

Creating a NIM model

```
SELECT aidb.create_model(
  my_nim_completions,
  nim_completions,
```

(continues on next page)

(continued from previous page)

```
{ "model": "meta/llama-3.2-1b-instruct" }::JSONB,
credentials=>{ "api_key": "sk-abc123xyz456def789ghi012jkl345mn"::JSONB};
```

Model configuration settings

The following configuration settings are available for OpenAI models:

- `model` — The model to use.
- `url` — The URL of the model to use. This setting is optional and can be used to specify a custom model URL.
 - * If `openai_completions` (or `completions`) is the model, `url` defaults to `https://api.openai.com/v1/chat/completions`.
 - * If `nim_completions` is the model, `url` defaults to `https://integrate.api.nvidia.com/v1/chat/completions`.
- `max_concurrent_requests` — The maximum number of concurrent requests to make to the OpenAI model. The default is 25.
- `max_tokens` — The maximum number of tokens generated by the model. The default is `NULL`. * If set to `NULL`, the maximum tokens parameter is omitted, and the model defaults to its internal maximum.

Maximum tokens

The `max_tokens` parameter sets the number of tokens the model is allowed to generate. This operates as an upper bound, allowing manage token budget constraints.

Parameters

- `size` — The maximum number of tokens generated by the model. * `format` — Defines how the maximum tokens parameter is sent to the model.
 - * If set to `default`, the `max_completion_tokens` parameter is provided in the request payload. This is ideal for models provided by OpenAI.
 - * If set to `legacy`, the `max_tokens` parameter is provided in the request payload. This is more common among non-OpenAI models.
 - * If set to `both`, both `max_completion_tokens` and `max_tokens` are provided in the request payload.

Example Usage

```
SELECT aadb.create_model(
  my_completions_model,
  completions,
  { "model": "meta/llama-3.2-1b-instruct", "max_tokens": { "format": "default",
  ↳ "size": 1024 } }::JSONB
```

Model credentials

The following credentials may be required by the service providing these models. Note: `api_key` and `basic_auth` are exclusive. You can use only one of these two options.

- `api_key` — The API key to use for Bearer Token authentication. The `api_key` is sent in a header field as `Authorization: Bearer <api_key>`.
- `basic_auth` — Credentials for HTTP Basic authentication. The credentials provided here are sent verbatim as `Authorization: Basic <basic_auth>`. If your server requires username/password with HTTP Basic authentication, use the syntax `username:password`. If only a token is required, provide only the token.

Embeddings

Model name: `embeddings`

Model aliases:

- `openai_embeddings`
- `nim_embeddings`

About embeddings

OpenAI embeddings is a text-embedding model that enables use of any OpenAI API-compatible text-embedding model. It's suitable for text classification, clustering, and other text-embedding tasks.

Based on the name of the model, the model provider sets defaults accordingly:

- When invoked as `embeddings` or `openai_embeddings`, the model provider defaults to using the OpenAI API.
- When invoked as `nim_embeddings`, the model provider defaults to using the NVIDIA NIM API.

Supported aidb operations

- `encode_text`
- `encode_text_batch`

Supported models

- Any text-embedding model that's supported by the provider.

Supported OpenAI models

- Any text-embedding model that's supported by OpenAI. This includes `text-embedding-3-small`, and `text-embedding-ada-002`. See a list of supported OpenAI models [here](#).
- The default is `text-embedding-3-small`.

Supported NIM models

- `nvidia/nv-embedqa-e5-v5` (default)

Creating the default with OpenAI model

```
SELECT aidb.create_model(my_openai_embeddings,
                        openai_embeddings,
                        credentials=>{"api_key": "sk-
→abc123xyz456def789ghi012jkl345mn": :JSONB);
```

As this example is defaulting the model to `text-embedding-3-small`, you don't need to specify the model in the configuration. But you do need to pass an OpenAI API key in the credentials. For that, you must pass credentials as a named parameter.

Creating a specific model

You can create any supported OpenAI embedding model using the `aidb.create_model` function. This example creates a `text-embedding-3-small` model with the name `my_openai_model`:

```
SELECT aidb.create_model(
  my_openai_model,
  openai_embeddings,
  {"model": "text-embedding-3-small"}::JSONB,
  {"api_key": "sk-abc123xyz456def789ghi012jkl345mn"}::JSONB
);
```

Because this example is passing the configuration options and the credentials, unlike the previous example, you don't need to pass the credentials as a named parameter.

Model configuration settings

The following configuration settings are available for OpenAI models:

- `model` — The OpenAI model to use.
- `url` — The URL of the model to use. This value is optional and can be used to specify a custom model URL. * If `openai_completions` (or `completions`) is the model, `url` defaults to `https://api.openai.com/v1/chat/completions`. * If `nim_completions` is the model, `url` defaults to `https://integrate.api.nvidia.com/v1/chat/completions`.
- `max_concurrent_requests` — The maximum number of concurrent requests to make to the OpenAI model. The default is 25.
- `max_batch_size` — The maximum number of records to send to the model in a single request. The default is 50.000.

Batch and parallel processing

The model providers for `embeddings`, `openai_embeddings`, and `nim_embeddings` support sending batch requests as well as concurrent requests. The settings `max_concurrent_requests` and `max_batch_size` control this behavior. When a model provider receives a set of records (for example, from a knowledge base pipeline) the following happens:

- Assume the knowledge base pipeline is configured with batch size 10.000. The model provider is configured with `max_batch_size=1000` and `max_concurrent_requests=5`.
- The provider collects up to 1000 records and sends them in a single request to the model.
- It sends 5 such large requests concurrently, until no more input records are left.
- In this example, the provider needs to send/receive 10 batches in total. * After sending the first 5, it waits for the responses to return. * Once a response is received, another request can be sent. * This means the provider won't wait for all 5 to return before sending off the next 5. Instead, it always keeps up to 5 requests in flight.

Note

The settings `max_concurrent_requests` and `max_batch_size` can have a significant impact on model performance. But they highly depend on the hardware and infrastructure.

We recommend leaving the defaults in place and [Pipelines knowledge base performance tuning](#)

The default `max_batch_size` of 50.000 is intentionally high to allow the pipeline to control the actual size of the batches.

The following credentials may be required by the service providing these models. Note: `api_key` and `basic_auth` are exclusive. You can use only one of these two options:

- `api_key` — The API key to use for Bearer Token authentication. The `api_key` will be sent in a header field as `Authorization: Bearer <api_key>`.
- `basic_auth` — credentials for HTTP Basic authentication. The credentials provided here will be sent verbatim as `Authorization: Basic <basic_auth>`. If your server requires username/password with HTTP Basic authentication, use the syntax `username:password`. If only a token is required, only provide the token.

NIM_CLIP

Model name: `nim_clip`

About CLIP

CLIP (Contrastive Language-Image Pre-training) is a model that learns visual concepts from natural language supervision. It is a zero-shot learning model that can be used for a wide range of vision and language tasks.

This specific model runs on NVIDIA NIM. More information about CLIP on NIM can be found [here](#).

Supported aidb operations

- `encode_text`
- `encode_text_batch`
- `encode_image`
- `encode_image_batch`

Supported models

build.nvidia.com

- `nvidia/nvclip` (default)

Creating the default model

```
SELECT aidb.create_model(
    my_nim_clip_model,
    nim_clip,
    credentials=>{"api_key": "<API_KEY_HERE>"}::JSONB
);
```

There's only one model, the default `nvidia/nvclip`, so you don't need to specify the model in the configuration.

Model configuration settings

The following configuration settings are available for CLIP models:

- `model` — The NIM model to use. The default is `nvidia/nvclip` and is the only model available.
- `url` — The default is `https://integrate.api.nvidia.com/v1/embeddings`.

Model credentials

The following credentials are required if executing inside build.nvidia.com:

- `api_key` — The build.nvidia.com API key to use for authentication.

NIM OCR

Model name: `nim_paddle_ocr`

About NVIDIA NIM Image OCR

The [NVIDIA NIM for Image OCR API](#) is an (Optical Character Recognition) microservice and interface that extracts text from images on NVIDIA NIM.

Supported aidb operations

- `perform_ocr`

Supported models

NVIDIA NIM for Image OCR

- `baidu/paddleocr` (default)

Creating the default model

```
SELECT aidb.create_model(
  my_paddle_ocr_model,
  nim_paddle_ocr,
  credentials => {"api_key": "__NVIDIA_NIM_API_KEY__"}::JSONB
);
```

There is currently only one model supported by [NVIDIA NIM for Image OCR API](#), the default `baidu/paddleocr`, so you don't need to specify the model in the configuration.

Model configuration settings

The following configuration settings are available for PADDLE_OCR models:

- `url` — The default is `https://ai.api.nvidia.com/v1/cv/baidu/paddleocr`.

Model credentials

The following credentials are required if executing inside build.nvidia.com:

- `api_key` — The build.nvidia.com API key to use for authentication.

Reranking (NIM)

Model name: `nim_reranking`

About reranking

Reranking is a method in text search that sorts results by relevance to make them more accurate. It gives scores to documents using cross-attention mechanisms, improving the initial search results.

Supported aidb operations

- rerank_text

Supported models

build.nvidia.com

- nvidia/llama-3.2-nv-rerankqa-1b-v2 (default)

Example

The function accepts a string as the “rerank query” and an array of texts to rerank. The `id` column in the output refers to the index of the text in the input array.

```
SELECT * from aidb.rerank_text(my_nim_reranker, how can I open a door?, [{"Ask for help,
→ Push the handle, Lie down and wait, Shout at it"}::text[]]) ORDER BY logit_score_
→DESC;
__OUTPUT__
      text      | logit_score | id
-----+-----+---
Push the handle | -3.697265625 | 1
Ask for help    |  -6.2578125 | 0
Shout at it     | -7.39453125 | 3
Lie down and wait |  -11.375 | 2
(4 rows)
```

Creating the default model

```
SELECT aidb.create_model(
  my_nim_reranker,
  nim_reranking,
  [{"url": "http://nim-nv-rerankqa-llama-1-1xgpu-g6-predictor.default.svc.cluster.
→local/v1/ranking", "model": "nvidia/llama-3.2-nv-rerankqa-1b-v2"}]
);
```

This example uses a locally deployed NIM model that does not require credentials. Credentials and other configuration can be provided as described in [Using models in AI Accelerator Pipelines](#).

Model configuration settings

The following configuration settings are available for NIM models:

- `model` — The NIM model to use. The default is `nvidia/llama-3.2-nv-rerankqa-1b-v2` and is the only supported model.
- `url` — The URL of the model to use. This setting is optional and can be used to specify a custom model URL. The default is `https://ai.api.nvidia.com/v1/retrieval`.
- `tls_config` — Optional TLS configuration for secure HTTPS endpoints. This field accepts an object with:

- `insecure_skip_verify` (boolean): If `true` , disables certificate verification (useful for development or self-signed certs).
- `ca_path` (string): Path to a PEM-encoded certificate authority file to trust when validating HTTPS connections.

The following example creates the model without using certificates (disabled)

```
SELECT aidb.create_model(
  my_nim_reranker_notsecure,
  nim_reranking,
  {
    "model": "nvidia/llama-3.2-nv-rerankqa-1b-v2",
    "url": "https://localhost/v1/ranking",
    "tls_config": {
      "insecure_skip_verify": true
    }
  }::jsonb
);
```

The following example shows how to create a model using certificates:

```
SELECT aidb.create_model(
  my_nim_reranker,
  nim_reranking,
  {
    "model": "nvidia/llama-3.2-nv-rerankqa-1b-v2",
    "url": "https://my.secure.endpoint/v1/ranking",
    "tls_config": {
      "ca_path": "/etc/aidb/myCA.pem"
    }
  }::jsonb
);
```

Model credentials

The following credentials are required if executing inside `build.nvidia.com`:

- `api_key` — The NVIDIA Cloud API key to use for authentication.

Model Serving Quickstart

Model Serving Quickstart

This page helps you quickly understand how to start using **Model Serving** within the AI Factory and where to find supporting documentation.

Model Serving in AI Factory enables you to deploy AI models (such as NVIDIA NIM Microservices) as scalable, production-grade inference services. It is powered by **KServe**, a Kubernetes-native model serving engine.

Where to start

1. Learn the concepts

Before deploying models, it's useful to understand how Model Serving works and how it fits into the AI Factory ecosystem:

- *Model Serving Explained*

- *AI Factory terminology*

2. Understand how AI Factory integrates Model Serving

Model Serving interacts with:

- **Model Library:** Browse and manage model images for deployment (coming soon)
- **Knowledge Bases (AIDB):** Vector stores that may use embedding models served by Model Serving
- **Gen AI Builder:** Applications may call into Model Serving endpoints for inferencing

3. Follow the How-To Guides

If you're ready to deploy or manage models:

- *How-To Deploy AI Models from Model Library*

Getting started checklist

Use this checklist to guide your progress depending on your experience level.

For new users (101 level)

- Read the *Model Serving Explained*
- Review key *AI Factory terminology*
- Understand how KServe powers Model Serving in the *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- Understand *Model Library Reference Manual* (coming soon)

Getting Started - AI Factory 101

For existing users familiar with Kubernetes (101 level)

- Verify your Kubernetes access in your HCP project
- Review the *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)* and *AI Factory terminology*
- Prepare your cluster prerequisites:
- GPU node pools (if needed)
- NVIDIA device plugin (if needed)
- Access to your container registry for model images
- Configure basic KServe resources:
- *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe*
- *How-To Configure a ClusterServingRuntime*
- *How-To Create an InferenceService for NVIDIA NIM Container*

Getting Started - AI Factory 101

For advanced users (201 level)

- Tune deployed InferenceService resource usage:
- *How-To Update GPU Resources for an InferenceService*
- Monitor deployed models:
- *How-To Monitor deployed models with KServe*
- *How-To Monitor deployed models with KServe* (coming soon)
- Understand traffic routing, canary rollouts, and scaling:
- *Gen AI Builder Concepts (Assistants, Knowledge, Tools, Threads)*
- Future: Advanced How-Tos

Building Production-Ready AI Features - AI Factory 201

For expert users (301 level)

- Manage your own custom model images
- Build and configure custom ServingRuntime definitions
- Use Transformers and Explainers in KServe (coming soon)
- Build CI/CD pipelines for deploying models in KServe
- Instrument InferenceServices for advanced observability

Advanced AI Factory Usage and Extensibility - AI Factory 301

Next steps

- *Model Serving Explained*
 - *How-To Deploy AI Models from Model Library*
 - *Model Library Reference Manual*
-

Use this quickstart as your launch point into Model Serving within AI Factory.

Using models with...

This section describes some particular ways to use models with AI Accelerator Pipelines. These techniques show how to use API-compatible services, running locally or in the cloud, with Pipelines and how to make use of NVIDIA NIM.

- *Using an OpenAI-compatible API with Pipelines* with Pipelines.
- *How-To Deploy an NVIDIA NIM Microservices available on build.nvidia.com/container with KServe* with Pipelines. * *Using NVIDIA NIM Microservices (Hosted)*
- *Using NVIDIA NIM Microservices available on build.nvidia.com in your environment*

Using an OpenAI-compatible API with Pipelines

To make use of an OpenAI-compliant API, you can use the embeddings or completions model providers. A knowledge base needs to encode first, so you can only use the embeddings model provider with a knowledge base.

Why use an OpenAI-compatible API?

Some examples of why you might want to use an OpenAI-compatible API include:

- If you have a local system running *Ollama* and you want that local system to handle embeddings. This assumes you've configured *Ollama as a server*.
- If you have access to a service that provides different or specifically tuned models, you can use it instead of other models.

Creating the model

The starting point for this process is creating a model. When you create a model, you can pass `options` and `credentials` to the registration. The defaults point to the OpenAI service endpoint. By overriding the defaults, you can point to any service.

This example creates a model that uses a local Ollama server:

```
select aidb.create_model(
my_local_ollama,
embeddings,
{"model": "llama3.2", "url": "http://llama.local:11434/v1/embeddings"}::JSONB);
```

Model name and model provider

The model name is the first parameter. For the example, it's set to `my_local_ollama`.

Specify the model provider as `embeddings`, which is the provider that defaults to using OpenAI servers. You can use the configuration parameter, to override this value to talk to any compliant server.

Configuration

The next parameter is the configuration. This is a JSON string. When expanded, it has two parameters: the model and the url.

```
{"model": "llama3.2", "url": "http://llama.local:11434/v1/embeddings"}::JSONB
```

In this case, we are setting the model to *“llama3.2”*, a relatively new and powerful model. Remember to run `ollama run llama3.2` to pull and start the model on the server.

The next JSON setting is the important one, overriding the endpoint that the aidb model will use. In this example:

- The server is running on a machine called `llama.local`.
- It has port 11434 (the default port for Ollama) open to service requests over HTTP (not HTTPS in this case).
- The path to the endpoint on the server is `/v1/embeddings`, which is the same as OpenAI.

Putting those components together we get `http://llama.local:11434/v1/embeddings` as our end point.

That completes the configuration parameter.

If the endpoint requires an API key, that would be passed in the `credentials` parameter. As this is a local model, we don't need to pass any credentials.

!!!Note Note When using indexing in pgvector, consider the [pgvector indexing limitations](#) . Aidb does not automatically configure an index today but if you add one manually, make sure it supports the number of dimensions your model uses.

Use the model name you created earlier to use the model just like any other Pipelines model. This example shows how to use the model to get an embedding:

```
select aidb.encode_text(my_local_ollama, I like it);
```

Pipelines takes care of all the connection management, freeing you to focus on your data and the model results.

Using NVIDIA NIM Microservices

You can use NVIDIA NIM Microservices available on build.nvidia.com models with AI Accelerator. The models can run in the build.nvidia.com or in your environment under your control.

You can learn how to use NVIDIA NIM with AI Accelerator in both scenarios:

- [Using NVIDIA NIM Microservices \(Hosted\)](#)
- [Using NVIDIA NIM Microservices available on build.nvidia.com in your environment](#)

Using NVIDIA NIM Microservices (Hosted)

To use a NVIDIA NIM that's hosted in NVIDIA's build.nvidia.com, you first need to select a model to use. This tutorial uses the NVIDIA NIM model `llama-3.3-70b-instruct`.

Prerequisites

- An build.nvidia.com account. (If you don't have one, you can create one [here](#) .)

Configuring the build.nvidia.com

1. Select a model

Choose a model from [NVIDIA's model library](#) . This example uses the `llama-3.3-70b-instruct` model.

2. Generate an API Key

You can create API keys from the model's page.

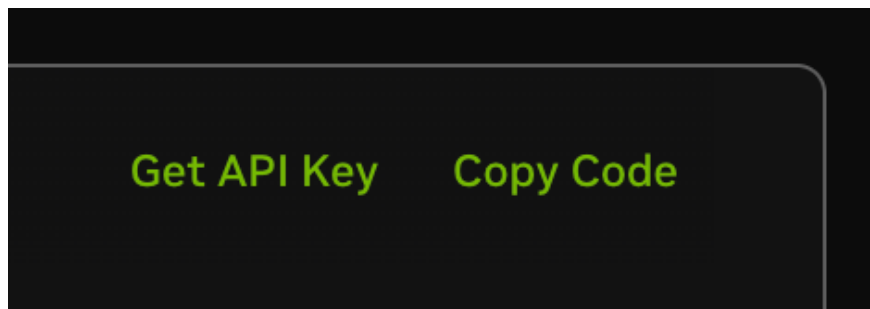


Fig. 1: Get API Key

To generate an API key, select **Get API Key**. If you haven't logged in, you're prompted to log in. If you haven't signed up, you're invited to do so. You'll receive an API key that you can use to interact with the model. This example refers

to this key as the <NIM API KEY> .

Integrating the model with AI Accelerator

1. Enable AI Accelerator in EDB Postgres AI

With EDB Postgres Advanced Server, EDB Postgres Extended Server, or a community Postgres instance running, connect to the database and enable the AI Accelerator extension:

```
CREATE EXTENSION aidb CASCADE;
```

2. Register the model

```
SELECT aidb.create_model(
  my_nim_llm,
  nim_completions,
  {
    "model": "meta/llama-3.3-70b-instruct",
    "api_key": "<NIM API KEY>"
  }::JSONB
);
```

3. Run the model

To interact with the model, execute the following query:

```
SELECT aidb.decode_text(my_nim_llm, Tell me a short, one sentence story);
__OUTPUT__

          decode_text
-----
As the clock struck midnight, a single tear fell from the porcelain dolls glassy
eye.
```

Your output may vary. You've successfully used NVIDIA NIM, integrated with AI Accelerator.

Using NVIDIA NIM Microservices available on build.nvidia.com in your environment

EDB AI Factory deploys [NVIDIA NIM microservices](#) with **KServe** using Kubernetes manifests. This guide provides ready-to-use YAML templates for the following NIM types:

- Chat Completion (LLM)
- Text Embedding
- Text Reranker
- Image Embedding
- Image OCR

The process is:

1. Set up NVIDIA NGC account and API key.
2. Create Kubernetes secrets for NGC access and image pulls.
3. Create GPU Nodes
4. Deploy an **InferenceService** for each model instance.

5. Validate
6. Use in AIDB

Supported Models and GPU Requirements

Model Type	NIM Model	NVIDIA Documented Resource Requirements
Text Completion	llama-3.3-70b-instruct	4 × L40S
Text Embeddings	arctic-embed-l	1 × L40S
Image Embeddings	nvclip	1 × L40S
OCR	paddleocr	1 × L40S
Text Reranking	llama-3.2-nv-rerankqa-1b-v2	1 × L40S

1. Set Up NVIDIA NGC API Key

1. Go to build.nvidia.com and log in with your NVIDIA account.
2. In the top-right menu, select **Setup** → **API Key**.
3. Click **Generate API Key**.
4. Copy the key and store it securely. You will need it for the Kubernetes secrets.

2. Create Kubernetes Secrets

Create the NGC API key secret:

```
kubectl create secret generic nvidia-nim-secrets \
  --from-literal=NGC_API_KEY="<YOUR_NGC_API_KEY>"
```

3. Create GPU Nodes

Ensure your Kubernetes cluster has nodes that satisfy the minimum GPU requirement for your chosen NIM model(s).

Node label:

```
nvidia.com/gpu: "true"
```

Node taint:

```
Key: nvidia.com/gpu
Value: "true"
Effect: NoSchedule
```

These settings ensure workloads requiring GPUs are scheduled correctly and only on GPU-enabled nodes.

4. Deploy the following KServe resources

Download the YAML file:

- Chat Completion Runtime: *llm-runtime.yaml*
- Chat Completion Service: *llm-service.yaml*
- Text Embedding Runtime: *image-embed-runtime.yaml*
- Text Embedding Service: *image-embed-service.yaml*
- Text Reranker Runtime: *rerank-runtime.yaml*
- Text Reranker Service: *rerank-service.yaml*
- Image Embedding Runtime: *image-embed-runtime.yaml*
- Image Embedding Service: *image-embed-service.yaml*
- Image OCR Runtime: *ocr-runtime.yaml*
- Image OCR Service: *ocr-service.yaml*

If needed: Update `<your-storage-class>`, and any resource settings to match your environment.

Apply to your cluster:

```
kubectl apply -f llm-runtime.yaml
kubectl apply -f llm-service.yaml
```

Check deployment status:

```
kubectl get clusterservingruntime
kubectl get inferencesservice
```

Repeat for each model type you want to deploy. **Verify endpoints** using the provided curl commands. **Integrate with EDB Postgres AI Accelerator** using the SQL commands in the registration section.

5. Validation

Find the endpoint:

```
kubectl get inferencesservice llama33-8b-instruct -o jsonpath={.status.url}
```

Test it:

```
curl -X POST "<ENDPOINT>/v1/chat/completions" \
-H "Content-Type: application/json" \
-d {"model": "meta/llama3-8b-instruct", "messages": [{"role": "user", "content":
↪ "Tell me a story"}]}
```

6. Use the Model

1. Enable AIDB

```
CREATE EXTENSION aidb CASCADE;
```

2. Register the model

```
SELECT aidb.create_model(
my_nim_llm,
nim_completions,
{"model": "meta/llama-3.3-70b-instruct", "url": "http://<ENDPOINT>:8000/v1/chat/
  ↳ completions"}::JSONB
);
```

3. Run the model

To interact with the model, execute the following query:

```
SELECT aidb.decode_text(my_nim_llm, Tell me a short, one sentence story);
__OUTPUT__

          decode_text
-----
↳ -----
As the clock struck midnight, a single tear fell from the porcelain dolls glassy
↳ eye.
```

Your output may vary. You've successfully used NVIDIA NIM Microservices via the EDB AI Accelerator.

Sources

- NVIDIA NIM KServe deployment resources: <https://catalog.ngc.nvidia.com/orgs/nim/resources/llm-nim-kserve>
- NVIDIA NIM API reference: <https://docs.nvidia.com/nim/large-language-models/latest/api-reference.html>
- KServe documentation: <https://kserve.github.io/website/latest/model-serving/serving-runtimes/>
- EDB AI Factory deployment examples: *Deploy NIM containers*
- Model examples and tags: build.nvidia.com model cards

embedding-runtime.yaml

```
apiVersion: serving.kservice.io/v1alpha1
kind: ClusterServingRuntime
metadata:
  name: nvidia-nim-snowflake-arctic-embed-1
  namespace: default
spec:
  containers:
    - env:
      - name: NIM_CACHE_PATH
        value: /tmp
      - name: NGC_API_KEY
```

(continues on next page)

(continued from previous page)

```

    valueFrom:
      secretKeyRef:
        name: nvidia-nim-secrets
        key: NGC_API_KEY
  image: upmdev.azurecr.io/nim/snowflake/arctic-embed-l:1.0.1
  name: kserve-container
  ports:
    - containerPort: 8000
      protocol: TCP
  resources:
    limits:
      cpu: "12"
      memory: 48Gi
    requests:
      cpu: "12"
      memory: 48Gi
  volumeMounts:
    - mountPath: /dev/shm
      name: dshm
  imagePullSecrets:
    - name: edb-cred
  protocolVersions:
    - v2
    - grpc-v2
  supportedModelFormats:
    - autoSelect: true
      name: nvidia-nim-snowflake-arctic-embed-l
      priority: 1
      version: "1.0.1"
  volumes:
    - emptyDir:
        medium: Memory
        sizeLimit: 40Gi
      name: dshm

```

embedding-service.yaml

```

apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  annotations:
    serving.kserve.io/enable-prometheus-scraping: "true"
    prometheus.kserve.io/port: "8000"
    prometheus.kserve.io/path: "/v1/metrics"
  name: nim-snowflake-arctic-embed-l
  namespace: default
spec:
  predictor:
    tolerations:
      - key: nvidia.com/gpu
        operator: Exists

```

(continues on next page)

(continued from previous page)

```

    effect: NoSchedule
nodeSelector:
  nvidia.com/gpu: "true"
imagePullSecrets:
  - name: edb-cred
model:
  modelFormat:
    name: nvidia-nim-snowflake-arctic-embed-l
  resources:
    limits:
      nvidia.com/gpu: "1"
    requests:
      nvidia.com/gpu: "1"
  runtime: nvidia-nim-snowflake-arctic-embed-l

```

image-embed-runtime.yaml

```

apiVersion: serving.kserve.io/v1alpha1
kind: ClusterServingRuntime
metadata:
  name: nvidia-nim-nvclip
  namespace: default
spec:
  containers:
    - env:
        - name: NIM_CACHE_PATH
          value: /tmp
        - name: NGC_API_KEY
          valueFrom:
            secretKeyRef:
              name: nvidia-nim-secrets
              key: NGC_API_KEY
      image: upmdev.azurecr.io/nim/nvidia/nvclip:latest
      name: kserve-container
      ports:
        - containerPort: 8000
          protocol: TCP
      resources:
        limits:
          cpu: "8"
          memory: 24Gi
        requests:
          cpu: "8"
          memory: 24Gi
      volumeMounts:
        - mountPath: /dev/shm
          name: dshm
    imagePullSecrets:
      - name: edb-cred
  protocolVersions:
    - v2

```

(continues on next page)

(continued from previous page)

```

- grpc-v2
supportedModelFormats:
- autoSelect: true
  name: nvidia-nim-nvclip
  priority: 1
  version: "latest"
volumes:
- emptyDir:
    medium: Memory
    sizeLimit: 20Gi
  name: dshm

```

image-embed-service.yaml

```

apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  annotations:
    serving.kserve.io/enable-prometheus-scraping: "true"
    prometheus.kserve.io/port: "8000"
    prometheus.kserve.io/path: "/v1/metrics"
  name: nim-nvclip
  namespace: default
spec:
  predictor:
    tolerations:
      - key: nvidia.com/gpu
        operator: Exists
        effect: NoSchedule
    nodeSelector:
      nvidia.com/gpu: "true"
    imagePullSecrets:
      - name: edb-cred
  model:
    modelFormat:
      name: nvidia-nim-nvclip
    resources:
      limits:
        nvidia.com/gpu: "1"
      requests:
        nvidia.com/gpu: "1"
    runtime: nvidia-nim-nvclip

```

llm-runtime.yaml

```

apiVersion: serving.kserve.io/v1alpha1
kind: ClusterServingRuntime
metadata:
  name: meta-nim-llama-3.3-70b-instruct
  namespace: default

```

(continues on next page)

(continued from previous page)

```

spec:
  containers:
    - env:
      - name: NIM_CACHE_PATH
        value: /tmp
      - name: NGC_API_KEY
        valueFrom:
          secretKeyRef:
            name: nvidia-nim-secrets
            key: NGC_API_KEY
      image: upmdev.azurecr.io/nim/meta/llama-3.3-70b-instruct:1.8.4
      name: kserve-container
      ports:
        - containerPort: 8000
          protocol: TCP
      resources:
        limits:
          cpu: "24"
          memory: 600Gi
        requests:
          cpu: "24"
          memory: 600Gi
      volumeMounts:
        - mountPath: /dev/shm
          name: dshm
      imagePullSecrets:
        - name: edb-cred
      protocolVersions:
        - v2
        - grpc-v2
      supportedModelFormats:
        - autoSelect: true
          name: meta-nim-llama-3.3-70b-instruct
          priority: 1
          version: "1.8.4"
      volumes:
        - emptyDir:
            medium: Memory
            sizeLimit: 60Gi
          name: dshm

```

llm-service.yaml

```

apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  annotations:
    serving.kserve.io/enable-prometheus-scraping: "true"
    prometheus.kserve.io/port: "8000"
    prometheus.kserve.io/path: "/v1/metrics"
  name: meta-nim-llama-3-3-70b-instruct

```

(continues on next page)

(continued from previous page)

```

namespace: default
spec:
  predictor:
    tolerations:
      - key: nvidia.com/gpu
        operator: Exists
        effect: NoSchedule
    nodeSelector:
      nvidia.com/gpu: "true"
    imagePullSecrets:
      - name: edb-cred
  model:
    modelFormat:
      name: meta-nim-llama-3.3-70b-instruct
    resources:
      limits:
        nvidia.com/gpu: "4"
      requests:
        nvidia.com/gpu: "4"
    runtime: meta-nim-llama-3.3-70b-instruct

```

ocr-runtime.yaml

```

apiVersion: serving.kserve.io/v1alpha1
kind: ClusterServingRuntime
metadata:
  name: nvidia-nim-baidu-paddleocr
  namespace: default
spec:
  containers:
    - env:
        - name: NIM_CACHE_PATH
          value: /tmp
        - name: NGC_API_KEY
          valueFrom:
            secretKeyRef:
              name: nvidia-nim-secrets
              key: NGC_API_KEY
      image: upmdev.azurecr.io/nim/baidu/paddleocr:latest
      name: kserve-container
      ports:
        - containerPort: 8000
          protocol: TCP
      resources:
        limits:
          cpu: "8"
          memory: 64Gi
        requests:
          cpu: "8"
          memory: 64Gi
      volumeMounts:

```

(continues on next page)

(continued from previous page)

```

    - mountPath: /dev/shm
      name: dshm
imagePullSecrets:
  - name: edb-cred
protocolVersions:
  - v2
  - grpc-v2
supportedModelFormats:
  - autoSelect: true
    name: nvidia-nim-baidu-paddleocr
    priority: 1
    version: "latest"
volumes:
  - emptyDir:
      medium: Memory
      sizeLimit: 12Gi
    name: dshm

```

ocr-service.yaml

```

apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  annotations:
    serving.kserve.io/enable-prometheus-scraping: "true"
    prometheus.kserve.io/port: "8000"
    prometheus.kserve.io/path: "/v1/metrics"
  name: nim-baidu-paddleocr
  namespace: default
spec:
  predictor:
    tolerations:
      - key: nvidia.com/gpu
        operator: Exists
        effect: NoSchedule
    nodeSelector:
      nvidia.com/gpu: "true"
    imagePullSecrets:
      - name: edb-cred
    model:
      modelFormat:
        name: nvidia-nim-baidu-paddleocr
      resources:
        limits:
          nvidia.com/gpu: "1"
        requests:
          nvidia.com/gpu: "1"
      runtime: nvidia-nim-baidu-paddleocr

```

rerank-runtime.yaml

```

apiVersion: serving.kserve.io/v1alpha1
kind: ClusterServingRuntime
metadata:
  name: nvidia-nim-nv-rerankqa-llama
  namespace: default
spec:
  containers:
    - env:
        - name: NIM_CACHE_PATH
          value: /tmp
        - name: NGC_API_KEY
          valueFrom:
            secretKeyRef:
              name: nvidia-nim-secrets
              key: NGC_API_KEY
      image: upmdev.azurecr.io/nim/nvidia/llama-3.2-nv-rerankqa-1b-v2:latest
      name: kserve-container
      ports:
        - containerPort: 8000
          protocol: TCP
      resources:
        limits:
          cpu: "4"
          memory: 12Gi
        requests:
          cpu: "4"
          memory: 12Gi
      volumeMounts:
        - mountPath: /dev/shm
          name: dshm
  imagePullSecrets:
    - name: edb-cred
  protocolVersions:
    - v2
    - grpc-v2
  supportedModelFormats:
    - autoSelect: true
      name: nvidia-nim-nv-rerankqa-llama
      priority: 1
      version: "latest"
  volumes:
    - emptyDir:
        medium: Memory
        sizeLimit: 20Gi
      name: dshm

```

rerank-service.yaml

```

apiVersion: serving.kserve.io/v1beta1
kind: InferenceService

```

(continues on next page)

(continued from previous page)

```

metadata:
  annotations:
    serving.kserve.io/enable-prometheus-scraping: "true"
    prometheus.kserve.io/port: "8000"
    prometheus.kserve.io/path: "/v1/metrics"
  name: nim-nv-rerankqa-llama-1
  namespace: default
spec:
  predictor:
    tolerations:
      - key: nvidia.com/gpu
        operator: Exists
        effect: NoSchedule
    nodeSelector:
      nvidia.com/gpu: "true"
    imagePullSecrets:
      - name: edb-cred
  model:
    modelFormat:
      name: nvidia-nim-nv-rerankqa-llama
    resources:
      limits:
        nvidia.com/gpu: "1"
      requests:
        nvidia.com/gpu: "1"
    runtime: nvidia-nim-nv-rerankqa-llama

```

Storage and access

Pipelines PGFS

Pipelines has a Postgres File System (PGFS) extension that allows you to access external storage. In particular, it allows you to store and retrieve files from either a public or private S3-compatible bucket or a local file system. The aidb extension addresses external storage mapped through PGFS as *storage locations* and abstracts away the details of the underlying storage.

PGFS is a key component of Pipelines, allowing you to work with data that isn't stored in the database, either because it's too large or because it isn't structured in a way that's suitable for a relational database.

PGFS is implemented as a foreign data wrapper (FDW) and installed as an extension. The extension provides a set of functions to interact with the file system from the database.

PGFS storage locations and AIDB volumes

To use a PGFS storage location with AIDB, a `volume` must be created in AIDB:

- **PGFS storage location:** Represents an external storage provider with all necessary configuration like a path and credentials.
- **AIDB volume:** Connects a storage location to AIDB. Multiple volumes for the same storage location are possible. In a volume, you configure the data format in the destination and optionally, a sub-path.

Installing the PGFS extension

See *Installing the AI Database extension* to learn how to install the PGFS extension as part of the Pipelines installation.

Using PGFS

See the *PGFS management functions* reference for details on the functions available in the PGFS extension. With these functions you can create and manage storage locations, to be later used through AIDB volumes.

PGFS functions for Pipelines

Creating a storage location

Start with creating a storage location. A storage location is a reference to a location in an external file system. You can create a storage location with the `pgfs.create_storage_location` function:

```
select pgfs.create_storage_location(name=>storage_location_name,
                                   url=>protocol://path);
```

You can also specify additional options and credentials for the storage location:

```
select pgfs.create_storage_location(name=>storage_location_name,
                                   url=>protocol://path,
                                   options => {},
                                   credentials => {});
```

Where:

- `name` is the name of the storage location.
- `url` is the URL of the storage location, which can be an S3-compatible storage location or a local file system path.
- `options` is an optional JSON object with additional options for the storage location.
- `credentials` is an optional JSON object with credentials for the storage location, if needed.

Note that there is also a `pgfs.create_storage_location` function with a `uuid` as the third parameter for a managed storage location id. The `uuid` is optional, and if not specified, set to null.

Storage provider types

Detailed instructions for the supported storage providers can be found here:

- *Pipelines PGFS with S3-compatible storage*
- *Pipelines PGFS with local file storage*

Creating a storage location with options and credentials

Using the `options` and `credentials` parameters allows a range of settings to be passed.

The `options` parameter is a JSON object that can be used to pass additional options to the storage location. The `credentials` parameter is a JSON object that can be used to pass credentials to the storage location.

The difference between `options` and `credentials` is that `options` remain visible to users querying the extension while `credentials` are hidden to all users except superusers and the user that creates the storage location.

Testing storage locations and using them with AIDB

To use a storage location with aidb, you need to create a volume from the storage location. To do that, see [AIDB volumes for accessing PGFS storage locations](#).

Listing storage locations

You can list all storage locations with the `pgfs.list_storage_locations` function:

```
select * from pgfs.list_storage_locations();
```

This command returns a table of currently defined storage locations. Credentials are shown only if the user has the necessary permissions. Otherwise the column is NULL.

Getting a storage location

You can get the details of a specific storage location with the `pgfs.get_storage_location` function:

```
select * from pgfs.get_storage_location(my_storage);
```

This command returns the details of the storage location named `my_storage`.

Updating a storage location

You can update a storage location with the `pgfs.update_storage_location` function:

```
select pgfs.update_storage_location(my_storage, s3://my_bucket, null, {"region": "eu-
->west"});
```

Deleting a storage location

You can delete a storage location with the `pgfs.delete_storage_location` function:

```
select pgfs.delete_storage_location(my_storage);
```

This command removes the storage location named `my_storage` from the database.

Pipelines PGFS with Google Cloud Storage

Overview: Google Cloud Storage

PGFS uses the `gs:` prefix to indicate a Google Cloud Storage bucket.

The general syntax for using GCS is:

```
select pgfs.create_storage_location(
    storage_location_name,
    gs://bucket_name,
    credentials => {}
);
```

credentials

The `credentials` argument is optional. It takes a value in JSON format and offers the following settings.

Option	Description
<code>google_application_credentials</code>	Path to the application credentials file
<code>google_service_account_key_file</code>	Path to the service account key file

See the [Google Cloud documentation](#) for more information on how to manage service account keys.

You can also set up these options via the equivalent environment variables to facilitate authentication in managed environments such as Google Kubernetes Engine.

Example: private GCS bucket

```
SELECT pgfs.create_storage_location(edb_ai_example_images, gs://my-company-ai-images,
                                credentials => {"google_service_account_key_file"
→": "/var/run/gcs.json"}
);
```

Example: authentication in GKE

Ensure that the `GOOGLE_APPLICATION_CREDENTIALS` or the `GOOGLE_SERVICE_ACCOUNT_KEY_FILE` environment variable is set on your PostgreSQL pod. Then, PGFS picks them up.

```
SELECT pgfs.create_storage_location(edb_ai_example_images, gs://my-company-ai-images);
```

Pipelines PGFS with local file storage

Overview: local file systems

PGFS uses the `file:` prefix to indicate a local file system.

The general syntax for using local file systems is this:

```
select pgfs.create_storage_location(
    local_images,
    file:///var/lib/edb/pipelines/images/
);
```

Note

Paths must always be absolute; e.g., start at the root `/`. Together with the protocol prefix `file://`, this means your path will have three slashes as in the example above.

Note

Any local path that you want to access must be allowlisted in the *PGFS settings*.

Note

Make sure to end your path with a / as in the examples above. This is required for path concatenation and validation with AIDB volumes.

Pipelines PGFS with S3-compatible storage

Overview: S3-compatible storage

PGFS uses the `s3:` prefix to indicate an S3-compatible storage system.

The general syntax for using S3 is:

```
select pgfs.create_storage_location(
    storage_location_name,
    s3://bucket_name,
    options => {},
    credentials => {}
);
```

Settings for the `options` argument in JSON format settings

Option	Description
<i>bucket</i>	The bucket name. Used to explicitly provide the bucket name if it can't be passed in the url.
<i>skip_signature</i>	Disable HMAC authentication. (Set this to "true" when you're not providing <code>access_key_id</code> / <code>secret_access_key</code> in the credentials.)
<i>region</i>	The region of the S3-compatible storage system. If the region isn't specified, the client attempts auto-discovery.
<i>endpoint</i>	The endpoint of the S3-compatible storage system.
<i>allow_http</i>	Whether the endpoint uses plain HTTP (rather than HTTPS/TLS). Set this to <i>true</i> if your endpoint starts with <i>http://</i> .

Settings for the `credentials` argument in JSON format

Option	Description
<i>access_key_id</i>	HMAC credentials (often the "username" in non-AWS S3 providers)
<i>secret_access_key</i>	HMAC credentials (often the "password" in non-AWS S3 providers)
<i>session_token</i>	A session token that can be used instead of HMAC credentials

Example: AWS S3 public bucket

This example uses a public bucket on AWS S3. Public buckets don't require authentication.

```
SELECT pgfs.create_storage_location(edb_ai_example_images, s3://public-ai-team,
    options => {"region": "eu-central-1", "skip_
    signature": "true"}
);
```

Example: AWS S3 private bucket

This example uses a private bucket on AWS S3. Private buckets require authentication. The example uses HMAC credentials.

```
SELECT pgfs.create_storage_location(internal_ai_project, s3://my-company-ai-images,
                                options => {"region": "eu-central-1"},
                                credentials => {"access_key_id": "secret",
→ "secret_access_key": "secret!"}
                                );
```

Example: non-AWS S3 / S3-compatible with HTTPS

This example uses an S3-compatible system like minIO. The `endpoint` must be provided in this case. You can omit it only when using AWS S3.

```
SELECT pgfs.create_storage_location(ai_images_local_minio, s3://my-ai-images,
                                options => {"endpoint": "https://minio-api.apps.
→ local"},
                                credentials => {"access_key_id": "my_username",
→ "secret_access_key": "my_password"}
                                );
```

Example: non-AWS S3 / S3-compatible with HTTP

This example uses an S3-compatible system like minIO. The `endpoint` must be provided in this case. You can omit it only when using AWS S3.

In this case, the server doesn't use TLS encryption, so the code configures a plain HTTP connection.

```
SELECT pgfs.create_storage_location(ai_images_local_minio,
                                s3://my-ai-images,
                                options => {"endpoint": "http://minio-api.apps.
→ local", "allow_http": "true"},
                                credentials => {"access_key_id": "my_username",
→ "secret_access_key": "my_password"}
                                );
```

PGFS settings

About PGFS settings

The PGFS extension has settings which control how the extension accesses data. These settings are GUCs (Grand Unified Configuration), which can be set using the Postgres `SET` command.

PGFS settings

pgfs.allowed_local_fs_paths

By default, PGFS is only able to access files in `/tmp/pgfs` in the local filesystem. This is a security measure to prevent the extension from accessing files it shouldn't.

If you want to allow PGFS to access files in other locations, you need to add those locations to a GUC setting, `pgfs.allowed_local_fs_paths`. This is a colon-separated list of paths that PGFS is allowed to access.

<code>pgfs.allowed_local_fs_paths</code> value	Description
<code>'/tmp/pgfs'</code>	The default location for PGFS to access files.
<code>'/tmp/pgfs:/tmp/other'</code>	PGFS is able to access files in both <code>/tmp/pgfs</code> and <code>/tmp/other</code> .
<code>'/'</code>	PGFS is able to access all files on the system. This is not recommended.
<code>''</code>	PGFS is not able to access any local files.

Use the Postgres `SET` command to change the setting. For example, to allow PGFS to access files in `/tmp/pgfs` and `/tmp/other`, you would run:

```
SET pgfs.allowed_local_fs_paths = /tmp/pgfs:/tmp/other;
```

AIDB volumes for accessing PGFS storage locations

Overview

Volumes in AIDB allow you to use external storage, instead of a postgres table, as the data source for pipelines.

Unstructured data like PDFs or images is often stored in object stores like AWS S3. With AIDB volumes, you can integrate such data into your AI pipelines.

PGFS storage locations and AIDB volumes

AIDB volumes allow access to PGFS storage locations. This means a storage location in PGFS must be created first. See [Pipelines PGFS reference](#).

- **PGFS storage location:** Represents an external storage provider with all necessary configuration like a path and credentials.
- **AIDB volume:** Connects a storage location to AIDB. Multiple volumes for the same storage location are possible. In a volume, you configure the data format in the destination and optionally, a sub-path.

Creating a volume

Assign a name to the new volume and specify which PGFS storage location it should point to.

The third argument is an optional sub-path in the storage location. This allows you to create multiple volumes in the same storage location, pointing to different paths within a bucket.

The last argument is the data format, the supported types are [aidb.PipelineDataFormat](#). This is metadata used by pipelines to decide how to treat the objects in the volume. The data format is not a filter. You should ensure that the volume only contains objects of the specified format.

```
SELECT aidb.create_volume(volume_name, pgfs_storage_location_name, /, Image);
```

Examples

You can find more information on [Pipelines PGFS reference](#).

Accessing a public S3 bucket with text documents:

```
SELECT pgfs.create_storage_location(
    aidb_rag_bucket, s3://aidb-rag-app,
    options => '{"region": "eu-central-1", "skip_signature": "true"}');
```

(continues on next page)

(continued from previous page)

```
SELECT aidb.create_volume(aidb_rag_bucket_volume, aidb_rag_bucket, /, Text);

SELECT * FROM aidb.list_volume_content(aidb_rag_bucket_volume);
```

Accessing a local directory containing images (e.g. for an OCR preparer pipeline). The volume points to the directory /tmp/pgfs/ocr_input/ :

```
CREATE EXTENSION IF NOT EXISTS pgfs;
SET pgfs.allowed_local_fs_paths = /tmp/pgfs;

SELECT pgfs.create_storage_location(local_tmp_pgfs, file:///tmp/pgfs/);

SELECT aidb.create_volume(ocr_input_volume, local_tmp_pgfs, ocr_input/, Image);

SELECT * FROM aidb.list_volume_content(ocr_input_volume);
```

Other management functions

The following management functions allow you to list and delete volumes:

```
SELECT aidb.list_volumes();
```

```
SELECT aidb.delete_volume(volume_name);
```

Note: deleting a PGFS storage location will delete all volumes created on top of it.

Accessing volumes

Volumes can be accessed in two different ways:

1. By using them within a pipeline as the data source
2. With direct access SQL functions

The direct access SQL functions are useful to test newly created storage locations and volumes before configuring them in a pipeline. They can also be used to build custom SQL queries that work with external storage.

Listing content

This command will return a table listing all the objects present in the volume.

```
SELECT * FROM aidb.list_volume_content(volume_name);
```

Reading an object

This command will read the contents of an object and return them as a BYTEA type.

```
SELECT aidb.read_volume_file(volume_name, object_name.txt);
```

As an example, if your objects contain plain text, they can be converted like this:

```
SELECT convert_from(aidb.read_volume_file(my_text_files, hello_world.txt), utf8);
```

Pipelines reference

Models

Tables

- *aidb.model_providers*
- *aidb.models*

Model functions

- *aidb.create_model*
- *aidb.get_model*
- *aidb.delete_model*
- *aidb.list_hcp_models*
- *aidb.create_hcp_model*
- *aidb.sync_hcp_models*
- *aidb.encode_text*
- *aidb.encode_text_batch*
- *aidb.decode_text*
- *aidb.decode_text_batch*
- *aidb.encode_image*
- *aidb.rerank_text*

Preparers

Preparer views

- *aidb.preps*
- *aidb.preparers*

Preparer types

- *aidb.DataPreparationOperation*

Preparer functions

- *aidb.create_table_preparer*
- :ref:`aidb.create\_preparer\_for\_table` (deprecated) <*aidb.create_preparer_for_table* (deprecated)>` (deprecated)
- *aidb.create_volume_preparer*
- :ref:`aidb.create\_preparer\_for\_volume` (deprecated) <*aidb.create_preparer_for_volume* (deprecated)>` (deprecated)
- *aidb.bulk_data_preparation*
- *aidb.set_auto_preparer*

- `:ref:``aidb.set_preparer_auto_processing` (deprecated) <aidb.set_preparer_auto_processing (deprecated)>` (deprecated)`
- `aidb.delete_preparer`

Knowledge bases

Knowledge base tables

- `aidb.kbs`
- `aidb.knowledge_bases`

Knowledge base functions

- `aidb.create_table_knowledge_base`
- `aidb.create_volume_knowledge_base`
- `aidb.set_auto_knowledge_base`
- `aidb.bulk_embedding`
- `aidb.retrieve_key`
- `aidb.retrieve_text`
- `:ref:``aidb.delete_knowledge_base` <aidb.delete_knowledge_base>``
- `aidb.create_volume_knowledge_base`
- `aidb.delete_volume`

PGFS

PGFS functions

- `pgfs.create_foreign_table`
- `pgfs.create_storage_location`
- `pgfs.create_storage_location_with_foreign_table`
- `pgfs.delete_storage_location`
- `pgfs.list_storage_locations`
- `pgfs.get_storage_location`
- `pgfs.update_storage_location`
- `pgfs.set_default_storage_location`
- `pgfs.get_default_storage_location`

Retrievers (deprecated)

Retrievers functions (deprecated)

- `:ref:``aidb.create_retriever_for_table` (deprecated) <aidb.create_retriever_for_table (deprecated)>` (deprecated)`
- `:ref:``aidb.create_retriever_for_volume` (deprecated) <aidb.create_retriever_for_volume (deprecated)>` (deprecated)`

- :ref:`aidb.set\_retriever\_auto\_processing` (deprecated) <aidb.set_retriever_auto_processing (deprecated)>` (deprecated)
- :ref:`aidb.delete\_retriever` (deprecated) <aidb.delete_retriever (deprecated)>` (deprecated)

Pipelines models reference

This reference documentation for EDB Postgres AI - AI Accelerator Pipelines models includes information on the functions and views available in the aidb extension for working with [Pipelines models reference](#).

Tables

aidb.model_providers

The aidb.model_providers table stores information about the model providers that are available.

Column	Type	Description
server_name	name	Name for the model server
server_description	text	Description of the model server
server_options	text[]	Options for the model server

aidb.models

Returns a list of all models in the registry and their configured options, including predefined models and user-created models.

Column	Type	Description
name	text	User-defined name for the model
provider	text	Name of the model provider
options	jsonb	Optional configuration for the model provider

Example

```
SELECT * FROM aidb.models;
```

__OUTPUT__

```
name | provider | options
-----+-----+-----
bert | bert_local | {"config={}"}
```

```
clip | clip_local | {"config={}"}
```

```
t5 | t5_local | {"config={}"}
```

Functions

aidb.create_model

Creates a new model in the system by saving its name, provider, and optional configuration.

Parameters

Parameter	Type	Default	Description
<i>name</i>	text		User-defined name for the model.
<i>provider</i>	text		Name of the model provider (as found in aidb.model_providers).
<i>config</i>	jsonb	'{}':jsonb	Optional configuration for the model provider. May include model-specific parameters such as <i>model</i> , <i>url</i> , and TLS options (e.g., <i>tls_config</i>).
<i>credentials</i>	jsonb	'{}':jsonb	Optional credentials for the model provider.
<i>replace_credentials</i>	boolean	false	If true, replaces the credentials for the model provider. If false, the credentials aren't overwritten.

Example

```
SELECT aidb.create_model(
    name => my_t5::text,
    provider => t5_local::character_varying,
    config => {"param1": "value1", "param2": "value2"}::jsonb,
    credentials => {"token": "abcd"}::jsonb
);
```

or equivalently, using default values:

```
SELECT aidb.create_model(my_t5, t5_local);
```

or if updating the credentials of a model's provider, which has already been created.

```
SELECT aidb.create_model(
    name => my_t5::text,
    provider => t5_local::character_varying,
    config => {"param1": "value1", "param2": "value2"}::jsonb,
    credentials => {"token": "abcd"}::jsonb,
    replace_credentials => true
);
```

TLS Configuration (optional)

To securely connect to HTTPS-based model endpoints, the `config` object can include a `tls_config` field:

```
"tls_config": {
  "insecure_skip_verify": true, // (optional) disables certificate validation
  "ca_path": "/etc/aidb/myCA.pem" // (optional) path to a trusted CA certificate
}
```

aidb.get_model

Returns the configuration for a model in the registry.

Parameters

Parameter	Type	Default	Description
<i>model_name</i>	text		Name of the model

Returns

Column	Type	Description
<i>name</i>	text	User-defined name for the model
<i>provider</i>	text	Name of the model provider
<i>options</i>	jsonb	Optional configuration for the model provider

Example

```
SELECT * FROM aidb.get_model(t5);
__OUTPUT__
name | provider | options
-----+-----+-----
t5   | t5_local | {"config={}"}
```

(1 row)

aidb.delete_model

Deletes a model from the registry.

Parameters

Parameter	Type	Default	Description
<i>model_name</i>	text		Name of the model

Example

```
SELECT aidb.delete_model(t5);
__OUTPUT__
delete_model
-----
(t5,t5_local,"{"config={}"})
(1 row)
```

Returns

Column	Type	Description
<i>delete_model</i>	jsonb	The name, provider, and options of the deleted model

aidb.list_hcp_models

Gets models running on the Hybrid Manager.

Returns

Column	Type	Description
<i>name</i>	text	The name of the model instance running on the HCP
<i>url</i>	text	The API URL of the model running on the HCP
<i>model</i>	text	The name the model running on the HCP

Example

```
SELECT * FROM aidb.get_hcp_models();
```

name	model	url
llama-3-1-8b-instruct-1xgpu-g6	llama-3-1-8b-instruct-1xgpu-g6-predictor.default.svc.cluster.local	http://llama-3-1-8b-instruct-1xgpu-g6-meta/llama-3.1-8b-instruct
llama-3-2-nv-embedqa-1b-v2	llama-3-2-nv-embedqa-1b-v2-predictor.default.svc.cluster.local	http://llama-3-2-nv-embedqa-1b-v2-predictor.nvidia/llama-3.2-nv-embedqa-1b-v2
meta-nim-llama3-70b-instruct-8xgpu-g5	meta-nim-llama3-70b-instruct-8xgpu-g5-predictor.default.svc.cluster.local	http://meta-nim-llama3-70b-instruct-8xgpu-g5-meta/llama3-70b-instruct

(3 rows)

aidb.create_hcp_model

Creates a new model in the system by referencing a running instance in the HCP

Parameters

Parameter	Type	Default	Description
<i>name</i>	text		User-defined name of the model
<i>hcp_model_name</i>	text		Name of the model instance running on HCP

aidb.sync_hcp_models

Creates models in the HCP and sets `is_hcp_model=true`; deletes models with that setting if not found in the HCP.

Returns

Column	Type	Description
<i>status</i>	text	Synchronization status; either <i>deleted</i> , <i>unchanged</i> , <i>created</i> , or <i>skipped</i> .
<i>model</i>	text	The name the synchronized HCP model.

aidb.encode_text

Encodes text using a model, generating a vector representation of a given text input.

Parameters

Parameter	Type	Default	Description
<i>model_name</i>	text		Name of the model to encode with
<i>text</i>	text		Text to encode

aidb.encode_text_batch

Encodes a batch of text using a model, generating a vector representation of a given text input.

Parameters

Parameter	Type	Default	Description
<i>model_name</i>	text		Name of the model to encode with
<i>text</i>	text[]		Array of text to encode

aidb.decode_text

Decodes text using a model, generating a vector representation of a given text input.

Parameters

Parameter	Type	Default	Description
<i>model_name</i>	text		Name of the model to decode with
<i>text</i>	text		Text to decode

Returns

Column	Type	Description
<i>decode_text</i>	text	The decoded text

aidb.decode_text_batch

Decodes a batch of text using a model, generating a representation of a given text input.

Parameters

Parameter	Type	Default	Description
<i>model_name</i>	text		Name of the model to decode with
<i>text</i>	text[]		Array of text to decode

Returns

Column	Type	Description
<i>decode_text</i>	text	The decoded text

aidb.encode_image

Encodes an image using a model, generating a vector representation of a given image input.

Parameters

Parameter	Type	Default	Description
<i>model_name</i>	text		Name of the model to encode with
<i>image</i>	bytea		Image to encode

Returns

Column	Type	Description
<i>encode_image</i>	bytea	The encoded image

aidb.rerank_text

Reranks text using a model, generating a vector representation of a given text input.

Parameters

Parameter	Type	Default	Description
<i>model_name</i>	text		Name of the model to rerank with
<i>query</i>	text		Query based on which the input will be ranked
<i>input</i>	text[]	[]	Inputs to be ranked

Returns

Column	Type	Description
<i>text</i>	text	The text from "input"
<i>logit_score</i>	double precision	Score/rank of this text
<i>id</i>	int	index that the text had in the input array

Pipelines reference - Preparers

This reference documentation for Pipelines preparers includes information on the functions and views available in the `aidb` extension related to *Pipelines reference - Preparers*. See *Pipelines knowledge bases usage* and *Where to find SDK references and examples?* for more details.

Views

`aidb.preparers`

Also referenceable as `aidb.preps`, the `aidb.preparers` view contains information about the configured data preparers. It includes the following columns:

Column	Type	Description
<code>id</code>	INTEGER	
<code>name</code>	TEXT	Name of the preparer.
<code>operation</code>	<i><code>aidb.DataPreparationOperation</code></i>	The kind of processing step to perform.
<code>destination_schema</code>	TEXT	Schema of the destination table to store the output data.
<code>destination_table</code>	TEXT	Name of the destination table to store the output data.
<code>destination_key_column</code>	TEXT	Column of the destination table that references the key in source data.
<code>destination_data_column</code>	TEXT	Column of the destination table to store the output data.
<code>options</code>	JSONB	Configuration options for the data preparation operation. Uses the same API as the <i><code>**vector-native capabilities**</code></i> .
<code>source_type</code>	TEXT	Type of source data the preparer is working with. Can be either 'Table' or 'Volume'.
<code>source_schema</code>	TEXT	Schema of the table with the source data the preparer will process. Applies only to preparers of <i>Table</i> source type.
<code>source_table</code>	TEXT	Name of the table with the source data the preparer will process. Applies only to preparers of <i>Table</i> source type.
<code>source_data_column</code>	TEXT	Column in the source table with the source data the preparer will process. Applies only to preparers of <i>Table</i> source type.
<code>source_key_column</code>	TEXT	Name of the key column in the source table for reference with the output processed data. Applies only to preparers of <i>Table</i> source type.

continues on next page

Table 48 – continued from previous page

Column	Type	Description
source_volume_name	TEXT	Name of the volume to use as a data source. Applies only to preparers of <i>Volume</i> source type.

Types

aidb.DataPreparationOperation

The `aidb.DataPreparationOperation` type is an enum that represents the different types of preprocessing steps that can be performed:

- `ChunkText`
- `SummarizeText`
- `ParseHtml`
- `ParsePdf`
- `PerformOcr`

Functions

aidb.create_table_preparer

Creates a preparer with a source data table.

Parameters

Parameter	Type	Default	Description
name	TEXT	Required	Name of the preparer
operation	<code>aidb.DataPreparationOperation</code>	Required	Type of data preparation operation
source_table	TEXT	Required	Name of the source data table
source_data_column	TEXT	Required	Column in the source table containing the raw data
destination_table	TEXT	Required	Name of the destination table
destination_data_column	TEXT	Required	Column in the destination table for processed data
source_key_column	TEXT	'id'	Unique column in the source table to use as key to reference the rows.
destination_key_column	TEXT	'id'	Key column in the destination table that references the <i>source_key_column</i>
options	JSONB	'{}'::JSONB	Configuration options for the data preparation operation. Uses the same API as the <i>vector-native capabilities</i> .

aidb.create_preparer_for_table (deprecated)

Replaced by *aidb.create_table_preparer* with same arguments.

aidb.create_volume_preparer

Creates a preparer for a given PGFS volume.

Parameters

Parameter	Type	Default	Description
name	TEXT	Required	Name of the preparer
operation	<i>aidb.DataPreparationOperation</i>	Required	Type of data preparation operation
source_volume_name	TEXT	Required	Name of the source volume containing the raw data
destination_table	TEXT	Required	Name of the destination table
destination_data_column	TEXT	Required	Column in the destination table for processed data
destination_key_column	TEXT	'id'	Key column in the destination table that uniquely identifies the processed data
options	JSONB	'{}':JSONB	Configuration options for the data preparation operation. Uses the same API as the <i>**vector-native capabilities**</i> .
invoker_role	TEXT	NULL	Role owning the tables, pipelines, and background job execution.

aidb.create_preparer_for_volume (deprecated)

Replaced by *aidb.create_volume_preparer* with same arguments.

aidb.bulk_data_preparation

Executes the configured data preparation operation on all data from the specified preparer's source.

Parameters

Parameter	Type	Description
preparer_name	TEXT	Name of the preparer

aidb.set_auto_preparer

Sets the automatic processing mode for this preparer. This function is used to enable and disable automatic data preparation: `Live` mode enables the Postgres trigger-based automation and `Disabled` disables all automation.

Parameters

Parameter	Type	Default	Description
preparer_name	TEXT		Name of preparer
mode	<i>aidb.PipelineAuto-ProcessingMode</i>		Desired processing mode

Example

```
SELECT aidb.set_auto_preparer(test_preparer, Live);
SELECT aidb.set_auto_preparer(test_preparer, Disabled);
```

`aidb.set_preparer_auto_processing` (deprecated)

Replaced by `aidb.set_auto_preparer` with same arguments.

`aidb.delete_preparer`

Deletes the preparer's configuration.

Parameters

Parameter	Type	Description
preparer_name	TEXT	Name of preparer to delete

Note

This function doesn't delete the destination table or any data in it.

Pipelines knowledge bases reference

This reference documentation for Pipelines knowledge bases includes information on the functions and views available in the `aidb` extension related to *Knowledge Bases*.

Views

`aidb.knowledge_bases`

Also referenceable as `aidb.kbs`, the `aidb.knowledge_bases` view provides a list of all knowledge bases in the database. It includes information about the knowledge base name, the model used, and the source data type.

Column	Type	Description
id	integer	
name	text	Name of the knowledge base.
vector_schema	text	Schema for <code>vector_table</code> .

continues on next page

Table 54 – continued from previous page

Column	Type	Description
vector_table	text	Name of the table where the embeddings are stored. Gets newly created if it doesn't exist. Managed by aidb.
vector_key_column	text	Column to use to store the key that references the key in source data when computing embeddings. We recommend using the default and letting aidb manage this table.
vector_data_column	text	Column to store embeddings in. We recommend using the default and letting aidb manage this table.
model_name	text	Name of the model to use for embedding computation and retrievals.
topk	integer	Default number of results to return during a retrieve. Similar to LIMIT in SQL.
distance_operator	<i>aidb.DistanceOperator</i>	During retrieval, the vector operation to use to compare the vectors.
options	jsonb	Unused.
auto_processing	<i>aidb.PipelineAutoProcessing-Mode</i>	Auto-processing mode.
owner_role	text	The Postgres role who created this pipeline. Background auto-processing will run the pipeline as this role.
batch_size	integer	How many records to process concurrently when running this pipeline.
background_sync_interval	interval	Used for background auto-processing. This is the interval between pipeline executions.
source_type	text	Indicates whether this pipeline uses a table or volume as data source.
source_schema	text	Schema for source_table.
source_table	text	Name of the table used as input for the pipeline. Unused if the knowledge base uses a volume as source.
source_data_column	text	Column name in the source table that Pipelines computes embeddings for. This is also the column that's returned in retrieve operations.
source_data_format	<i>aidb.PipelineDataFormat</i>	Format of the data the knowledge base is working with. Uses type :ref:`aidb.PipelineDataFormat` <#aidbpipelinedataformat>`.
source_key_column	text	Column to use as key for storing the embedding in the vector table. This provides a reference from the embedding to the source data.
source_volume_name	text	Name of the volume to use as a data source. Only applicable to knowledge bases configured with aidb.create_volume_knowledge_base().

aidb.knowledge_base_stats

Also referenceable as `aidb.kbstat`, the `aidb.knowledge_base_stats` view provides current statistics about auto processing for knowledge base pipelines.

Column	Type	Description
knowledge base	text	Name of the knowledge base.
auto processing	<i>aidb.PipelineAutoProcessing-Mode</i>	Auto-processing mode.
table: unprocessed rows	bigint	For table knowledge bases: The number of unprocessed rows that are new or might have changed since the last execution.
volume: scans completed	bigint	For volume knowledge bases: The number of full listings of the source volume that were performed so far.
count(source records)	bigint	The number of records in the source of the knowledge base. For volume knowledge bases, this is number of records seen during the last full scan.
count(embeddings)	bigint	The number of embeddings stored in the destination table.

Types

aidb.DistanceOperator

The `aidb.DistanceOperator` type is an enum that represents the distance operators that can be used during retrieval.

Value	Description
L2	Euclidean distance
Inner	Inner product
Cosine	Cosine similarity
L1	L1 distance
Hamming	Hamming distance
Jaccard	Jaccard distance

SQL definition:

```
CREATE TYPE DistanceOperator AS ENUM (
  L2,
  InnerProduct,
  Cosine,
  L1,
  Hamming,
  Jaccard
);
```

aidb.PipelineDataFormat

The `aidb.PipelineDataFormat` type is an enum that represents the data formats that can be used as source data.

Value	Description
Text	Text data
Image	Image data
Pdf	PDF data

SQL definition:


```
CREATE TYPE PipelineDataFormat AS ENUM (
    Text,
    Image,
    Pdf
);
```

aidb.PipelineAutoProcessingMode

The `aidb.PipelineAutoProcessingMode` type is an enum used to define how auto processing for a Pipeline should behave (e.g. a knowledge base or a Preparer).

Value	Description
Live	New data is processed immediately while being added (using Postgres Triggers)
Background	Continuous processing in the background (using Postgres background workers)
Disabled	No automated processing

```
CREATE TYPE PipelineAutoProcessingMode AS ENUM (
    Live,
    Background,
    Disabled
);
```

Functions

aidb.create_table_knowledge_base

Creates a knowledge base for a given table.

Parameters

Parameter	Type	Default	Description
name	TEXT	Required	Name of the knowledge base.
model_name	TEXT	Required	Name of the model to use.
source_table	regclass	Required	Name of the table to use as source.
source_data_column	TEXT	Required	Column name in source table to use.
source_data_format	<i>aidb.PipelineDataFormat</i>	Required	Format of data in that column ("Text", "Image", "PDF").
source_key_column	TEXT	'id'	Unique column in the source table to use as key to reference the rows.
vector_table	TEXT	NULL	
vector_data_column	TEXT	'embeddings'	
vector_key_column	TEXT	'id'	
topk	INTEGER	1	
distance_operator	<i>aidb.DistanceOperator</i>	'L2'	
options	JSONB	'{}'::JSONB	Unused
auto_processing	<i>aidb.PipelineAutoProcessingMode</i>	'Disabled'	Configure auto-processing for this pipeline.

continues on next page

Table 59 – continued from previous page

Parameter	Type	Default	Description
index_type	TEXT	'vector'	Type of index to use for the vector table.
batch_size	int	100	How many records to process concurrently.
background_sync_interval	interval	'30 seconds'	Interval between pipeline executions if background auto-processing is configured.
invoker_role	TEXT	NULL	Role owning the tables, pipelines, and background job execution.

distance_operator	index_type
L2	<i>vector_l2_ops</i>
InnerProduct	<i>vector_ip_ops</i>
Cosine	<i>vector_cosine_ops</i>
L1	<i>vector_l1_ops</i>

- If `index_type` is set to `ivfflat`, the system will create a **IVFFlat** index on the vector table.
- If `index_type` is set to `disabled`, no index will be created.

```
SELECT aidb.create_table_knowledge_base(
    name => test_knowledge_base,
    model_name => bert,
    source_table => test_source_table,
    source_data_column => content,
    source_key_column => doc_id,
    source_data_format => Text
);
```

aidb.create_volume_knowledge_base

Creates a knowledge base for a given PGFS volume.

Parameters

Parameter	Type	Default	Description
name	TEXT	Required	Name of the knowledge base
model_name	TEXT	Required	Name of the model
source_volume_name	TEXT	Required	Name of the volume
vector_table	TEXT	NULL	Name of the vector table
vector_data_column	TEXT	'embeddings'	Name of the vector column
vector_key_column	TEXT	'id'	Name of the key column
topk	INTEGER	1	Number of results to return
distance_operator	aidb.distanceoperator	'L2'	Distance operator
options	JSONB	'{}'::JSONB	Options

continues on next page

Table 61 – continued from previous page

Parameter	Type	Default	Description
auto_processing	<i>aidb.PipelineAuto-ProcessingMode</i>	'Disabled'	Configure auto-processing for this pipeline.
index_type	TEXT	'vector'	Type of index to use for the vector table.
batch_size	int	100	How many records to process concurrently.
background_sync_interval	interval	'30 seconds'	Interval between pipeline executions if background auto-processing is configured.
invoker_role	TEXT	NULL	Role owning the tables, pipelines, and background job execution.

distance_operator	index_type
L2	<i>vector_l2_ops</i>
InnerProduct	<i>vector_ip_ops</i>
Cosine	<i>vector_cosine_ops</i>
L1	<i>vector_l1_ops</i>

- If `index_type` is set to `ivfflat`, the system will create a **IVFFlat** index on the vector table.
- If `index_type` is set to `disabled`, no index will be created.

```
SELECT aidb.create_volume_knowledge_base(
    name => demo_vol_knowledge_base,
    model_name => simple_model,
    source_volume_name => demo_bucket_vol
);
```

aidb.set_auto_knowledge_base

Sets the auto-processing mode for this knowledge base.

Parameters

Parameter	Type	Default	Description
knowledge_base_name	TEXT		Name of knowledge base for which to enable auto-processing
mode	<i>aidb.PipelineAuto-ProcessingMode</i>		Desired auto-processing mode
batch_size	INTEGER	DE-FAULT NULL	How many records to process in one batch in Disabled (aka. Manual) or Background processing
background_sync_interval	INTERVAL	DE-FAULT NULL	Desired sync interval for background processing

Example

```
SELECT aidb.set_auto_knowledge_base(test_knowledge_base, Live);
SELECT aidb.set_auto_knowledge_base(test_knowledge_base, Disabled, batch_size => 100);
SELECT aidb.set_auto_knowledge_base(test_knowledge_base, Background);
SELECT aidb.set_auto_knowledge_base(test_knowledge_base, Background, batch_size =>
↪100, background_sync_interval => 10 seconds);
```

aidb.bulk_embedding

Compute the embeddings for the source records in this pipeline and store them in vector destination.

The behavior of this function depends on the configured `auto_processing` mode:

- **Live and Disabled auto-processing:** Directly process all source records.
- **Background auto-processing:** Mark all source records for processing but don't perform the operation. The background worker will perform it.

Parameters

Parameter	Type	Default	Description
knowledge_base_name	TEXT		Name of knowledge base for which to generate embeddings.
silent	BOOL	false	Disable printing status and progress logs.

Example

```
aidb=# SELECT aidb.bulk_embedding(kb_volume_image_manual);
__OUTPUT__
INFO: kb_volume_image_manual: (re)setting state table to process all data...
INFO: kb_volume_image_manual: Starting... Batch size 10, count(known source
↪records): 0, scans completed: 0, count(embeddings): 0
INFO: kb_volume_image_manual: Batch iteration finished, count(known source records):
↪10, scans completed: 0, count(embeddings): 10
[...]
INFO: kb_volume_image_manual: Batch iteration finished, count(known source records):
↪177, scans completed: 0, count(embeddings): 177
INFO: kb_volume_image_manual: finished, count(known source records): 177, scans
↪completed: 1, count(embeddings): 177
```

aidb.retrieve_key

Retrieves a key from matching embeddings without looking up the source data.

Parameters

Parameter	Type	Default	Description
knowledge_base_name	TEXT		Name of knowledge base to use for retrieval
query_string	TEXT		Query string to use for retrieval
number_of_results	INTEGER	0	Number of results to return

Example

```
SELECT * FROM aidb.retrieve_key(test_knowledge_base, shoes, 2);
__OUTPUT__
key | distance
-----+-----
43941 | 0.2938963414490189
19337 | 0.3023805122617119
(2 rows)
```

aidb.retrieve_text

Retrieves the source text data from matching embeddings by joining the embeddings with the source table.

Parameters

Parameter	Type	Default	Description
knowledge_base_name	TEXT		Name of knowledge base to use for retrieval
query_string	TEXT		Query string to use for retrieval
number_of_results	INTEGER	0	Number of results to return

Returns

Column	Type	Description
<i>key</i>	text	Key of the retrieved data
<i>value</i>	text	Value of the retrieved data
<i>distance</i>	double precision	Distance of the retrieved data from the query

Example

```
SELECT * FROM aidb.retrieve_text(test_knowledge_base, jacket, 2);
__OUTPUT__
key | value | distance
-----+-----+-----
19337 | United Colors of Benetton Men Stripes Black Jacket | 0.2994317672742334
55018 | Lakme 3 in 1 Orchid Aqua Shine Lip Color | 0.3804609668507203
(2 rows)
```

aidb.delete_knowledge_base

Deletes only the knowledge base's configuration from the database.

Parameters

Parameter	Type	Default	Description
knowledge_base_name	TEXT		Name of knowledge base to delete

Example

```
select aidb.delete_knowledge_base(test_knowledge_base);
__OUTPUT__
delete_knowledge_base
-----
(1 row)
```

aidb.create_volume

Creates a volume from a PGFS storage location for use as a data source in knowledge bases.

Parameters

Parameter	Type	Default	Description
name	TEXT		Name of the volume to create
server_name	TEXT		Name of the storage location to use for the volume
path	TEXT		Path to the volume in the storage location
mime_type	TEXT		Type of data in the volume (Text or Image)

Example

```
select aidb.create_volume(demo_bucket_vol, demo_bucket, demo_bucket/demo_folder,
↪Text);
__OUTPUT__
create_volume
-----
(1 row)
```

aidb.list_volumes

Lists all the volumes that were created in the database.

Example

```
select * from aidb.list_volumes();
```

aidb.delete_volume

Deletes a volume from the database.

Parameters

Parameter	Type	Default	Description
volume_name	TEXT		Name of the volume to delete

Example

```
select aidb.delete_volume(demo_bucket_vol);
__OUTPUT__
delete_volume
-----
(1 row)
```

Pipelines PGFS reference

This reference documentation for EDB Postgres AI - AI Accelerator Pipelines PGFS includes information on the functions and views available in the *Pipelines PGFS reference* extension. These functions give AIDB access to S3-compatible file systems, Google Cloud Storage buckets, and local file systems.

pgfs

Functions

- *pgfs.create_foreign_table*
- *pgfs.create_storage_location*
- *pgfs.create_storage_location_with_foreign_table*
- *pgfs.delete_storage_location*
- *pgfs.list_storage_locations*
- *pgfs.get_storage_location*
- *pgfs.update_storage_location*

Functions

pgfs.create_foreign_table

Creates a foreign table in the database.

Parameters

Parameter	Type	Default	Description
<i>table_name</i>	text		
<i>server_name</i>	text		

Example

```
SELECT pgfs.create_foreign_table(my_table, my_server);
```

pgfs.create_storage_location

Creates a storage location in the database.

Parameters

Parameter	Type	Default	Description
<i>name</i>	text		Name for storage location
<i>url</i>	text		URL for this storage location (prefix <i>s3:</i> , <i>gs:</i> , or <i>file:</i>)
<i>options</i>	json		Options for the storage location (optional)
<i>credentials</i>	json		Credentials for the storage location (optional)

OR

Parameter	Type	Default	Description
<i>name</i>	text		Name for storage location
<i>url</i>	text		URL for this storage location (prefix <i>s3:</i> , <i>gs:</i> , or <i>file:</i>)
<i>msl_id</i>	uuid		Unused
<i>options</i>	json		Options for the storage location (optional)
<i>credentials</i>	json		Credentials for the storage location (optional)

Example

```
SELECT pgfs.create_storage_location(my_storage,
                                   s3://my_bucket, 123e4567-e89b-12d3-a456-
↪ 426614174000,
                                   {},
                                   {});
```


pgfs.create_storage_location_with_foreign_table

Creates a storage location in the database and associates it with a foreign table.

Parameters

Parameter	Type	Default	Description
<i>storage_location_name</i>	text		Name for storage location
<i>url</i>	text		URL for this storage location (prefix <i>s3:</i> , <i>gs:</i> or <i>file:</i>)
<i>options</i>	json		Options for the storage location (optional)
<i>credentials</i>	json		Credentials for the storage location (optional)

OR

Parameter	Type	Default	Description
<i>storage_location_name</i>	text		Name for storage location
<i>url</i>	text		URL for this storage location (prefix <i>s3:</i> , <i>gs:</i> or <i>file:</i>)
<i>msl_id</i>	uuid		Unused
<i>options</i>	json		Options for the storage location (optional)
<i>credentials</i>	json		Credentials for the storage location (optional)

Example

```
SELECT pgfs.create_storage_location_with_foreign_table(my_storage,
                                                    s3://my_bucket, 123e4567-e89b-
↪ 12d3-a456-426614174000,
                                                    {},
                                                    {});
```

pgfs.delete_storage_location

Deletes a storage location from the database.

Parameters

Parameter	Type	Default	Description
<i>storage_location_name</i>	text		Name of storage location

Example

```
SELECT pgfs.delete_storage_location(my_storage);
```

pgfs.get_default_storage_location

Returns the default storage location.

Returns

Column	Type	Description
<i>default_storage_location</i>	text	

Example

```
SELECT * FROM pgfs.get_default_storage_location();
```

pgfs.get_storage_location

Returns information about a storage location.

Parameters

Parameter	Type	Default	Description
<i>storage_location_name</i>	text		Name of storage location

Returns

Column	Type	Description
<i>name</i>	text	
<i>url</i>	text	
<i>mst_id</i>	uuid	
<i>options</i>	json	
<i>credentials</i>	json	

Example

```
SELECT * FROM pgfs.get_storage_location(my_storage);
```

pgfs.list_storage_locations

Lists all storage locations in the database.

Returns

Column	Type	Description
<i>name</i>	text	
<i>url</i>	text	
<i>msl_id</i>	uuid	
<i>options</i>	json	
<i>credentials</i>	json	

Example

```
SELECT * FROM pgfs.list_storage_locations();
```

pgfs.set_default_storage_location

Sets the default storage location.

Parameters

Parameter	Type	Default	Description
<i>storage_location_name</i>	text		

Example

```
SELECT pgfs.set_default_storage_location(my_storage);
```

pgfs.update_storage_location

Updates a storage location in the database.

Parameters

Parameter	Type	Default	Description
<i>name</i>	text		
<i>url</i>	text		
<i>msl_id</i>	uuid		
<i>options</i>	json		
<i>credentials</i>	json		

Example

```
SELECT pgfs.update_storage_location(my_storage, s3://my_bucket, 123e4567-e89b-12d3-
↪a456-426614174000, {}, {});
```

Views

There are no user-facing views in the `pgfs` schema.

Tables

There are no user-facing tables in the `pgfs` schema.

Pipelines retrievers (deprecated) reference

Important

The use of the term "retrievers" has been deprecated in version 4.0 and later. It is replaced by the term "knowledge base" and users should consult the [Pipelines knowledge bases reference](#) reference content. For backwards compatibility, the retrievers API is still available, but it is recommended to use the knowledge base API for new implementations. This section covers the deprecated functions only. For the new API, see [Pipelines knowledge bases reference](#).

`aidb.RetrieverSourceDataFormat`

No longer available. Use `aidb.PipelineDataFormat` instead.

Functions

`aidb.create_retriever_for_table` (deprecated)

Replaced by `aidb.create_table_knowledge_base`

Creates a retriever for a given table.

Parameters

Parameter	Type	Default	Description
<code>name</code>	TEXT	Required	Name of the retriever
<code>model_name</code>	TEXT	Required	Name of the model to use
<code>source_table</code>	regclass	Required	Name of the table to use as source
<code>source_data_column</code>	TEXT	Required	Column name in source table to use
<code>source_data_type</code>	<code>aidb.Pipeline-DataFormat</code>	Required	Type of data in that column ("Text", "Image", "PDF")
<code>source_key_column</code>	TEXT	'id'	Column to use as key to reference the rows
<code>vector_table</code>	TEXT	NULL	
<code>vector_data_column</code>	TEXT	'embeddings'	

continues on next page

Table 83 – continued from previous page

Parameter	Type	Default	Description
vector_key_column	TEXT	'id'	
topk	INTEGER	1	
distance_operator	<i>aidb.DistanceOperator</i>	'L2'	
options	JSONB	'{}'::JSONB	Options
index_type	TEXT	'vector'	Type of index to use for the vector table.

distance_operator	index_type
L2	<i>vector_l2_ops</i>
InnerProduct	<i>vector_ip_ops</i>
Cosine	<i>vector_cosine_ops</i>
L1	<i>vector_l1_ops</i>

- If `index_type` is set to `ivfflat`, the system will create a **IVFFlat** index on the vector table.
- If `index_type` is set to `disabled`, no index will be created.

```
SELECT aidb.create_retriever_for_table(
    name => test_retriever,
    model_name => simple_model,
    source_table => test_source_table,
    source_data_column => content,
    source_data_type => Text,
);
```

`aidb.create_retriever_for_volume` (deprecated)

Replaced by `aidb.create_volume_knowledge_base`

Creates a retriever for a given PGFS volume.

Parameters

Parameter	Type	Default	Description
name	TEXT	Required	Name of the retriever
model_name	TEXT	Required	Name of the model
source_volume_name	TEXT	Required	Name of the volume
vector_table	TEXT	NULL	Name of the vector table
vector_data_column	TEXT	'embeddings'	Name of the vector column
vector_key_column	TEXT	'id'	Name of the key column
topk	INTEGER	1	Number of results to return
distance_operator	<code>aidb.distanceoperator</code>	'L2'	Distance operator
options	JSONB	'{}'::JSONB	Options
index_type	TEXT	'vector'	Type of index to use for the vector table.

distance_operator	index_type
L2	<i>vector_l2_ops</i>
InnerProduct	<i>vector_ip_ops</i>
Cosine	<i>vector_cosine_ops</i>
L1	<i>vector_l1_ops</i>

- If `index_type` is set to `ivfflat`, the system will create a [IVFFlat](#) index on the vector table.
- If `index_type` is set to `disabled`, no index will be created.

```
SELECT aidb.create_retriever_for_volume(
    name => demo_vol_retriever,
    model_name => simple_model,
    source_volume_name => demo_bucket_vol
);
```

`aidb.set_retriever_auto_processing` (deprecated)

Replaced by `aidb.set_auto_knowledge_base`

Sets the auto-processing mode for this knowledge base.

Parameters

Parameter	Type	Default	Description
<code>retriever_name</code>	TEXT		Name of retriever for which to enable auto-embedding
<code>mode</code>	<code>aidb.PipelineAuto-ProcessingMode</code>		Desired auto-processing mode

Example

```
SELECT aidb.set_retriever_auto_processing(test_retriever, Live);
SELECT aidb.set_retriever_auto_processing(test_retriever, Disabled);
```

`aidb.delete_retriever` (deprecated)

Replaced by `:ref:`aidb.delete_knowledge_base` <aidb.delete_knowledge_base>`

Deletes only the retriever's configuration from the database.

Parameters

Parameter	Type	Default	Description
<code>retriever_name</code>	TEXT		Name of retriever to delete

Example

```
select aidb.delete_retriever(test_retriever);
__OUTPUT__
delete_retriever
-----
(1 row)
```

EDB Postgres AI - AI Accelerator release notes

The EDB Postgres AI - AI Accelerator describes the latest version of AI Accelerator. The release notes provide information on what was new in each release. Differences from the technical preview are highlighted in the 1.0.7 release notes.

AI Accelerator version	Release Date
AI Accelerator - Pipelines 4.4.0 release notes	13 Aug 2025
AI Accelerator - Pipelines 4.3.0 release notes	04 Jul 2025
AI Accelerator - Pipelines 4.2.0 release notes	16 Jun 2025
AI Accelerator - Pipelines 4.1.1 release notes	28 May 2025
AI Accelerator - Pipelines 4.1.0 release notes	19 May 2025
AI Accelerator - Pipelines 4.0.1 release notes	09 May 2025
AI Accelerator - Pipelines 4.0.0 release notes	05 May 2025
AI Accelerator - Pipelines 3.0.1 release notes	03 Apr 2025
AI Accelerator - Pipelines 2.2.1 release notes	13 Mar 2025
AI Accelerator - Pipelines 2.1.2 release notes	24 Feb 2025
AI Accelerator - Pipelines 2.1.1 release notes	03 Feb 2025
AI Accelerator - Pipelines 2.0.0 release notes	13 Jan 2025
1.0.7	10 Dec 2024

AI Accelerator - Pipelines 4.4.0 release notes

Released: 13 August 2025

This release introduces the ability to use aidb without superuser privileges.

Highlights

- Introduced non-superuser execution support for enhanced security and access control.
- Improved performance when running knowledge base pipelines on large datasets.

Enhancements

Description	Addresses
Added support for non-super user - - - Now <i>aidb</i> can have non-superuser execution support for enhanced security and access control.	
Improved performance - - - Now <i>aidb</i> can have improved performance when running knowledge base pipelines on large datasets.	

AI Accelerator - Pipelines 4.3.0 release notes

Released: 4 July 2025

This is a minor release that includes bug fixes and small enhancements.

Highlights

- This is the first release that supports in-place update via `ALTER EXTENSION aidb UPDATE;` with all settings and pipelines remaining in place.
- Two new LLMs are now supported; “Llama instruct” as a new built-in model and external Google Gemini via a new model provider.

Enhancements

Description	Addresses
Added support for Google Gemini - Now <i>aidb</i> can access Gemini text models.	
Added support for local Llama - Now <i>aidb</i> can execute local Llama instruct models.	
Removed excessive log messages that caused performance impact. - The live processing modes on the knowledge base pipeline used to log each processed record. This is now no longer the case. Printing and streaming these logs often took more time than processing itself.	
Fixed a bug that prevented users from using certain strings as names for pipelines. - Strings with capital letters in them were not supported as names for pipelines. This is now fixed.	

AI Accelerator - Pipelines 4.2.0 release notes

Released: 16 June 2025

This is a minor release that includes bug fixes and small enhancements.

Highlights

- Add support for the `retrieve_text()` function when using volumes as data sources for knowledge base pipelines.
- Bug fix for local file processing in volume knowledge bases.
- Bug fix for easy model syncing in hybrid manager deployments.
- Add support for preparer pipeline volume destination in foreign schemas.

Enhancements

Description	Addresses
Add support for the <i>retrieve_text()</i> function when using volumes as data sources for knowledge base pipelines. The <i>retrieve_text()</i> function can now be used with knowledge bases that use volumes as data source. Previously, only <i>retrieve_key()</i> was supported and users had to manually load the content of the objects in they application.	
Fewer mandatory values when configuring preparer pipelines. Added default values for some preparer pipeline options to reduce the number of mandatory settings.	
Add support for preparer pipeline volume destination in foreign schemas. Volumes used as destinations for preparer pipelines can now be located in arbitrary schemas within the postgres database. This completes schema support for AIDB.	

Bug Fixes

Description	Addresses
Bug fix for local file processing in volume knowledge bases. An issue existed where a number of local files in a volume knowledge base were not processed due to an issue with the ordering of file results. This has been fixed.	

AI Accelerator - Pipelines 4.1.1 release notes

Released: 28 May 2025

This is a minor release that includes bug fixes. Note: this release depends on the new PGFS version 2.0.2. If you're using PGFS together with AIDB, please update PGFS as well.

Highlights

- Bug fixes and UX enhancements.

Enhancements

Description	Addresses
Change the tracking of "run IDs" for volume knowledge bases from a PG sequence to a PG table column - - - Knowledge bases with volume source need to assign an incrementing ID to each listing done on the source. This is necessary to detect when a listing is complete and recognize which previously seen objects were not encountered any more. In this release the method for tracking run IDs is changed to a table column since the Postgres sequence used before did not work on Postgres Distributed (PGD) clusters.	
Support AIDB volumes in arbitrary Postgres schemas - - - Volumes for AIDB can be created in arbitrary schemas in Postgres either by explicitly passing a schema to the create function, or by setting a "current schema" via the search path. When using volumes in AIDB pipelines, the volume reference needs to be stored. AIDB will now store and explicitly use a schema when later executing a pipeline. Previously, the search path was used to find volumes.	
Background worker dispatcher is no longer persistent - - - When background workers are enabled for AIDB, a root-worker will start a database dispatcher for each database within the Postgres instance. This database dispatcher checks if AIDB is installed in the DB and if any pipelines need to be run in the background. The database dispatcher is now no longer a permanent process by default. When a database does not use AIDB, the database dispatcher exits and will be re-started to check for AIDB every 2 minutes. This avoids holding a permanent connection to the DB that would block e.g. "drop table" commands. If AIDB is installed in the DB, then the database dispatcher worker will keep running. Users can drop the extension in order to release the connection.	

AI Accelerator - Pipelines 4.1.0 release notes

Released: 19 May 2025

This is a minor release that includes enhancements to the preparer pipeline and the model API providers.

Highlights

- Automatic unnesting of Preparer results for operations that transform the shape of data.
- Batch processing for embeddings with external models.

Enhancements

Description	Addresses
Automatic unnesting of Preparer results for operations that transform the shape of data. - - - The preparer pipeline for operations that transform the shape of their input data with an additional dimension now unnest their result collections. This allows the output of preparers to be consumed much more easily by other preparers or knowledge bases. Unnested results are returned with a new <i>part_id</i> column to track the new dimension. There is also a new <i>unique_id</i> column to uniquely identify the combination of the source key and <i>part_id</i> .	
Batch processing for embeddings with external models. - - - The external model providers <i>embeddings</i> , <i>openai_embeddings</i> , and <i>nim_embeddings</i> can now send a batch of inputs in a single request, rather than multiple concurrent requests. This can improve performance and hardware utilization. The feature is fully configurable and can also be disabled.	

continues on next page

Table 95 – continued from previous page

Description	Addresses
Change output column for <i>chunk_text()</i> primitive function. - - - The enumeration column returned by the <i>chunk_text()</i> primitive function is now <i>part_id</i> instead of <i>chunk_id</i> to match the other Preparer primitives/operations.	

AI Accelerator - Pipelines 4.0.1 release notes

Released: 9 May 2025

This is a minor release that includes a few bug fixes and enhancements to the knowledge base pipeline.

Highlights

- Bug fixes and performance enhancements.
- Simplified model integration for HCP.

Enhancements

Description	Addresses
Bug fixes for knowledge base and preparer pipeline schema handling. - - - The knowledge base and preparer pipeline now support arbitrary Postgres schemas for source and destination tables/volumes. A bug prevented users for configuring explicit schemas via Postgres qualified identifiers (<i>schema.name</i>) when referencing source or destination tables/volumes in the create pipeline calls. This bug is now fixed.	
Bug fix for knowledge base result accuracy. - - - A bug in the batch-processing code for knowledge base pipelines would, under certain circumstances, lead to inaccurate results during retrieval. This bug was introduced in the 4.0.0 release and is now fixed.	
Simplified model integration for HCP. - - - External models, running on the HCP Model Serving infrastructure, can now be listed and integrated into AIDB with new helper functions <i>aidb.get_hcp_models()</i> and <i>aidb.create_hcp_model()</i> .	
Performance enhancement for embeddings processing with external models. - - - When using external models for embeddings (e.g. with the <i>nim_embeddings</i> model adapter), AIDB performs additional API calls in order to probe the model service for the response type. These calls are now skipped in most situations to reduce the overhead to a minimum when running embedding processing. AIDB will now use actual results to determine the response type.	

AI Accelerator - Pipelines 4.0.0 release notes

Released: 5 May 2025

This release introduces a completely re-written processing back-end for the knowledge base pipeline. Retrievers are renamed to knowledge bases and some types and functions are also renamed for consistency. The preparer pipeline now supports OCR and can output its results to a volume. As always, bug fixes and other enhancements to existing functionality are also included.

Highlights

- New processing back-end for the knowledge base pipeline supporting: background processing, batch processing, full iterative sync for tables and volumes.
- Added OCR support to the preparer pipeline.
- Added the ability to write preparer pipeline output to a volume.
- Renaming of retrievers to knowledge bases.
- Renaming of preparer functions for consistency.
- Renaming of “processing mode” in pipelines config to “auto processing” for consistency.
- Added arm64 support for the AI Accelerator on Debian 12 and RHEL 9.
- Added support for writing preparer output to object storage volumes with configurable TLS and custom certificate validation.

Enhancements

Description	Addresses
Renamed retrievers to knowledge bases and renamed preparer functions. ----- The retriever pipeline has been renamed to the knowledge base pipeline. The functions <code>aidb.create_retriever_for_table()</code> and <code>aidb.create_retriever_for_volume()</code> have been renamed to <code>aidb.create_knowledge_base_for_table()</code> and <code>aidb.create_knowledge_base_for_volume()</code> , respectively. Also renamed is the <code>aidb.delete_retriever</code> function (to <code>aidb.delete_knowledge_base()</code>) and the <code>aidb.RetrieverSourceDataFormat</code> enum has been renamed to <code>aidb.PipelineDataFormat</code> . For consistency, the preparer pipeline functions have also been renamed. The <code>aidb.create_preparer_for_table()</code> function has been renamed to <code>aidb.create_table_preparer()</code> and <code>aidb.create_preparer_for_volume()</code> has been renamed to <code>aidb.create_volume_preparer()</code> . The <code>aidb.retrievers</code> view is now the <code>aidb.knowledge_bases</code> view. Aliases for the <code>aidb.knowledge_bases</code> and <code>aidb.preparers</code> views have also been added as <code>aidb.kbs</code> and <code>aidb.preps</code> respectively. The older functions and view names are still available but marked as deprecated. They will be removed in a future release.	
Renamed "processing mode" in pipelines config to "auto processing" for consistency ----- The setting for "auto processing" in pipeline creation, config and overview functions has been re-named from <code>processing_mode</code> to <code>auto_processing</code> . We did this to have consistency with the name of the feature itself which is called auto-processing.	
Added the ability to write preparer pipeline output to a volume. ----- Preparer pipelines now support AIDB volumes (e.g. an S3 bucket using PGFS) as output destination. Previously, only Postgres tables were supported. Volumes as input source were already supported together with tables, so now all 4 source/destination combinations are possible.	
Added arm64 support for the AI Accelerator on Debian 12 and RHEL 9. ----- The AI Accelerator now supports arm64 architecture on Debian 12 and RHEL 9. This includes support for the arm64 architecture in the <code>aidb</code> packages.	
The <code>aidb.rerank_text()</code> function now returns results as a rich table ----- The <code>aidb.rerank_text()</code> function used to rerank a list of texts based on a ranking query, now returns its results in a table, one row per text. The table includes columns for the raw <code>logit_score</code> , the text itself and the index from the input. This makes it much easier to use the result in larger SQL query e.g. by joining with other data or sorting the rows by <code>logit_score</code> .	

continues on next page

Table 97 – continued from previous page

Description	Addresses
Improved debugging capabilities for AI models accessed over HTTP APIs ----- AIDB now prints rich debugging output for any errors encountered when using external models via API. This includes NIM models and OpenAI API compatible models. The HTTP status code, error response content, and request content are logged to help users debug issues like incorrect endpoints, incorrect model configuration or incorrect credentials, among others.	
Added TLS configuration support for models using HTTP APIs. ----- Models that connect to external HTTP endpoints can now be configured with TLS settings via the <i>tls_config</i> object. This supports disabling certificate verification using <i>insecure_skip_verify</i> for development, or specifying a custom certificate authority using the <i>ca_path</i> field. These settings allow secure integration with services using self-signed or private CA certificates.	
Added PerformOcr operation to Preparer. ----- Preparer pipelines can now utilize a <i>PerformOcr</i> operation that leverages the PaddleOCR model with the NVIDIA NIM Image OCR API .	

AI Accelerator - Pipelines 3.0.1 release notes

Released: 3 April 2025

In this release, we're adding auto-processing support to the new preparer pipeline and make a breaking change to the return type of two batch processing functions.

As always, we also include enhancements to existing functionality.

Highlights

- The new Data Preparation Pipeline now supports Live auto processing when the data source is a table.
- The retriever pipeline now automatically creates vector indexes for faster lookups.
- Model encode/decode batch functions now return rows, making them fit more naturally into SQL queries.
- Ubuntu 24.04LTS builds are now available

Enhancements

Description	Addresses
Live auto processing for the preparer pipeline --- The preparer pipeline now support live auto processing for data sources that are postgres tables. When enabled, aidb will create triggers on the source table to automatically run the preparer task.	

continues on next page

Table 98 – continued from previous page

Description	Addresses
Model encode/decode batch functions now return rows instead of an array of results. --- The two functions <code>aidb.decode_text_batch()</code> and <code>aidb.encode_text_batch()</code> now return a set of rows as their result, instead of a single row containing an array. This change was made based on customer feedback. In the case of the encode function, the return type is a list of vector embeddings. These are arrays themselves (arrays of floating point numbers). By returning this set of vectors as an array, we end up with a two-dimensional array. These kinds of arrays are not well supported in Postgres and get flattened in the return type of the function. Complicated parsing of the flattened array was required to work with the return type. This was less of an issue with the decode function but we decided to change both to retain consistency in the return types. Now, the batch functions return a set of rows, one row for each input value.	
Auto-processing (auto-embedding) is now configured via new function. --- The two functions <code>aidb.enable_auto_embedding_for_table()</code> and <code>aidb.disable_auto_embedding_for_table()</code> have been marked as deprecated. The new function <code>aidb.set_retriever_auto_processing()</code> accepts a "processing mode" argument: <i>Live</i> and <i>Disabled</i> mode are used to turn auto-embedding on and off. For the preparer pipeline, the function <code>aidb.set_preparer_auto_processing()</code> serves the same purpose.	
The retriever pipeline now automatically creates vector indexes for faster lookups. --- Vector indexes are now automatically created when creating retrievers in the pipeline. These indexes can be configured in the <code>aidb.create_retriever_for_table()</code> and <code>aidb.create_retriever_for_volume()</code> functions which now accept an optional <code>index_type</code> argument. This allows you to specify the index type to be used for the retriever. The default value is <i>vector</i> .	
Added support for <code>pgfs.allowed_local_paths</code> GUC --- A new GUC <code>pgfs.allowed_local_paths</code> has been added to control the local file access in PGFS. By default, local file access is now restricted to <code>/tmp/pgfs</code> . This GUC can be used to allow access to other directories. See <i>**vector-native capabilities**</i> for more information.	

AI Accelerator - Pipelines 2.2.1 release notes

Released: 13 March 2025

In this release, we're introducing our second Pipeline Data Preparation, also known as "preparer." Our first Pipeline, the "Retriever," launched with the initial release of AIDB, enabling automated embedding computation, semantic search, and RAG support.

These predefined Pipelines are at the core of AIDB, providing end-to-end automation for common AI application workflows. With the addition of preparer, we expand this automation further—allowing data to be pre-processed and optimized for use in Pipelines like the Retriever.

As always, we also include enhancements to existing functionality.

Highlights

- The new Data Preparation Pipeline allows automated chunking, parsing and summarizing of data to make it ready for use in AI Applications.

Features

Description	Addresses
Introduced preparer, A New Pipeline for Data Preparation --- The new preparer pipeline supports automated chunking, parsing and summarizing of data to make it ready for use in AI Applications.	

Enhancements

Description	Addresses
Added support for HTTP basic authentication in OpenAI Model adapters --- The OpenAI Model Adapters (e.g. <i>completions</i> or <i>embeddings</i>) now support "HTTP Basic authentication" via the credential field <i>basic_auth</i> . Previously, only "Bearer authentication" via <i>api_key</i> was supported.	
Added a Postgresql view for configured Models --- The Models configured in AIDB can now be listed as a view <i>aidb.models</i> rather than needing a function call as before (<i>aidb.list_models()</i>).	
Added autodetection for Model dimensions when using external Models over an API --- Model dimensions for external models (via e.g. <i>completions</i> or <i>embeddings</i> model adapter) no longer need to be explicitly configured. AIDB will now auto-detect the number of dimensions of the returned embeddings/vectors.	

AI Accelerator - Pipelines 2.1.2 release notes

Released: 24 February 2025

In this maintenance release, we fix two bugs and improve the model provider listing output.

Highlights

- Model provider output now contains descriptions of the providers.

Enhancements

Description	Addresses
Added model provider descriptions - - - The model provider view <i>select * from aidb.model_providers</i> ; now includes a description column detailing the purpose and capabilities of each model provider.	

Bug Fixes

Description	Addresses
Fixed the volume listing command - - - The <i>aidb.list_volumes()</i> ; call is now functional. A bug prevented this call from succeeding (without affecting any other functionality).	

continues on next page

Table 102 – continued from previous page

Description	Addresses
Fixed the search path handling - - - Some internal commands used to append <i>aidb</i> to the search_path. This failed in certain cases depending on the contents of the existing search path. We changed the implementation to no longer modify the search_path in order to avoid this issue. This also means that the existing search_path no longer gets modified by AIDB.	
Credentials are now completely optional for OpenAI compatible models - - - Previously the <i>aidb.create_model()</i> function required a key with an empty value to be passed for the api-key. It now allows you to omit the <i>credentials</i> parameter.	

AI Accelerator - Pipelines 2.1.1 release notes

Released: 3 February 2025

In this release, we add support for Nvidia NIM, introduce new model names, and improve the retriever pipeline.

Highlights

- Support for Nvidia NIM added.
- `embeddings` and `completions` are new model names.
- Reranking using NIM is now available.
- Source tables in retriever pipelines now support schemas.

Enhancements

Description	Addresses
Renamed <code>openai_embedding</code> and <code>openai_completion</code> - - - The model names <i>openai_embedding</i> and <i>openai_completion</i> have been renamed to <i>embeddings</i> and <i>completions</i> respectively. The old names are still supported for backward compatibility.	
Improved model provider credential management - - - Improve model provider credential management in the <code>create_model()</code> call. New credentials no longer get silently ignored if credentials already exist.	
Added handler for the FDW (foreign data wrappers) used for the model registry - - - The handler has no functionality but it satisfies the FDW handler interface. This avoids certain issues with PG system calls that happen when FDWs don't have any handlers.	
Added schema support / fully qualified identifiers to the retriever pipeline. - - - For retrievers with table sources, the source table and vector table can now be in any schema in the PG database. The schema can either be passed during retriever creation (fully qualified identifier) or can be omitted. If the schema is omitted, we resolve and store the applicable schema.	

AI Accelerator - Pipelines 2.0.0 release notes

Released: 13 January 2025

Updating the GA release of EDB Postgres AI - AI Accelerator - Pipelines, in response to feedback.

Highlights

- Improved API naming in functions and arguments.

Enhancements

Description	Addresses
Renaming of functions/arguments - - - Management functions move to use <i>create</i> rather than <i>register</i> and <i>delete</i> rather than <i>drop</i> . Function argument names lose the preceding <i>p_</i> to be consistent.	

Legal

Open Source Licenses for software incorporated in EDB Postgres AI - AI Accelerator - Pipelines

License Summary

The following table lists the licenses for the software incorporated in EDB Postgres AI - AI Accelerator - Pipelines.

- MIT
- Apache 2.0
- Apache 2.0 with Exceptions
- BSD Two Clause
- BSD Three Clause
- ICSL
- OpenSSL
- zlib
- Mozilla Public License 2.0
- Unicode DFS 2016
- Unicode 3.0
- Univ of Illinois/NCSA

License Details by Package

- accelerate-src 0.3.2: <https://github.com/blas-lapack-rs/accelerate-src> : (MIT License OR Apache License 2.0)
- addr2line 0.24.2 : (MIT License OR Apache License 2.0)
- Adler2 2.0.0: <https://crates.io/crates/adler2> : (BSD Zero Clause License OR MIT License OR Apache License 2.0)
- alexcrichon/cfg-if 1.0.0: <https://github.com/alexcrichon/cfg-if> : (MIT License AND Apache License 2.- 0)
- alexcrichon/jobserver-rs 0.1.32: <https://github.com/alexcrichon/jobserver-rs> : Apache License 2.0
- alexcrichon/openssl-probe 0.1.5: <https://github.com/alexcrichon/openssl-probe> : (MIT License OR Apache License 2.0)
- alexcrichon/proc-macro2 1.0.89: <https://github.com/alexcrichon/proc-macro2> : (MIT License OR Apache License 2.0)

- alexcrichton/rustc-demangle 0.1.24: <https://github.com/alexcrichton/rustc-demangle> : (MIT License OR Apache License 2.0)
- alexcrichton/socket2-rs 0.5.7: <https://github.com/alexcrichton/socket2-rs> : (MIT License OR Apache License 2.0)
- aligned-vec 0.5.0: <https://crates.io/crates/aligned-vec> : MIT License
- allocator-api2 0.2.18: <https://github.com/zakarumych/allocator-api2> : (MIT License OR Apache License 2.0)
- android_system_properties 0.1.5: https://github.com/nical/android_system_properties : (MIT License OR Apache License 2.0)
- android-tzdata 0.1.1: <https://crates.io/crates/android-tzdata> : (MIT License OR Apache License 2.0)
- annotate-snippets 0.9.2: <https://crates.io/crates/annotate-snippets> : (MIT License OR Apache License 2.0)
- anstyle 1.0.10: <https://github.com/rust-cli/anstyle> : (MIT License OR Apache License 2.0)
- anyhow 1.0.93 : (MIT License OR Apache License 2.0)
- arbitrary 1.4.1 : (MIT License OR Apache License 2.0)
- arg_enum_proc_macro 0.3.4: https://crates.io/crates/arg_enum_proc_macro : MIT License
- assert-json-diff 2.0.2: <https://github.com/davidpdrsn/assert-json-diff> : MIT License
- async-trait 0.1.83 : (MIT License OR Apache License 2.0)
- atomic-traits 0.3.0: <https://github.com/flier/rust-atomic-traits> : (MIT License OR Apache License 2.0)
- atomic-waker 1.1.2: <https://github.com/stjepang/atomic-waker> : (MIT License OR Apache License 2.0)
- autocfg 1.4.0 : Apache License 2.0
- av1-grain 0.2.3: <https://github.com/rust-av/av1-grain> : BSD 2-clause “Simplified” License
- avif-serialize 0.8.2: <https://lib.rs/avif-serialize> : BSD 3-clause “New” or “Revised” License
- bit_field 0.10.2: https://crates.io/crates/bit_field : (MIT License OR Apache License 2.0)
- bitflags 1.3.2: <https://github.com/rust-lang-nursery/bitflags> : (MIT License OR Apache License 2.0)
- bitflags 2.6.0: <https://github.com/rust-lang-nursery/bitflags> : (MIT License OR Apache License 2.0)
- bitstream-io 2.6.0: <https://github.com/tufffy/bitstream-io> : (MIT License OR Apache License 2.0)
- bitvec 1.0.1: <https://myrlyn.net/crates/bitvec> : MIT License
- bit-vec 0.6.3: <https://github.com/contain-rs/bit-vec> : (MIT License OR Apache License 2.0)
- block-buffer 0.10.4 : (MIT License OR Apache License 2.0)
- bluss/arrayvec 0.7.6 : Apache License 2.0
- bluss/either 1.13.0 : Apache License 2.0
- bluss/indexmap 2.6.0: <https://docs.rs/indexmap/> : (MIT License AND Apache License 2.0)
- bluss/scopeguard 1.2.0 : Apache License 2.0
- briansmith/ring 0.17.8: <https://crates.io/crates/ring> : (MIT License AND ISC License AND The Open SSL License)
- briansmith/untrusted 0.9.0 : ISC License
- built 0.7.5 : MIT License
- BurntSushi/byteorder 1.5.0: <https://github.com/BurntSushi/byteorder> : The Unlicense

- BurntSushi/regex-automata 0.4.8: <https://github.com/BurntSushi/regex-automata> : (MIT License OR Apache License 2.0)
- BurntSushi/rust-memchr 2.7.4: <https://github.com/BurntSushi/rust-memchr> : (MIT License OR The Unlicense)
- BurntSushi/same-file 1.0.6: <https://github.com/BurntSushi/same-file> : (MIT License AND The Unlicense)
- BurntSushi/ucd-generate 0.1.7: <https://github.com/BurntSushi/ucd-generate> : Apache License 2.0
- BurntSushi/walkdir 2.5.0: <https://github.com/BurntSushi/walkdir> : The Unlicense
- bytemuck 1.19.0 : (zlib License OR MIT License OR Apache License 2.0)
- bytemuck_derive 1.8.0 : (zlib License OR MIT License OR Apache License 2.0)
- byteorder-lite 0.1.0: <https://github.com/image-rs/byteorder-lite> : (MIT License OR The Unlicense)
- camino 1.1.9: <https://crates.io/crates/camino> : (MIT License OR Apache License 2.0)
- candle-core 0.7.2 : (MIT License OR Apache License 2.0)
- candle-nn 0.7.2 : (MIT License OR Apache License 2.0)
- candle-transformers 0.7.2 : (MIT License OR Apache License 2.0)
- cargo_toml 0.19.2: https://lib.rs/cargo_toml : (MIT License OR Apache License 2.0)
- cavif (Rust) 0.11.11: <https://lib.rs/cavif> : BSD 3-clause “New” or “Revised” License
- cc-rs 1.1.36: <https://docs.rs/cc> : (MIT License OR Apache License 2.0)
- cee-scape 0.2.0: <https://github.com/pnkfelix/cee-scape> : (MIT License OR Apache License 2.0)
- cexpr 0.6.0: <https://crates.io/crates/cexpr> : (MIT License OR Apache License 2.0)
- cfg_aliases 0.2.1: https://github.com/katharostech/cfg_aliases : MIT License
- cfg-expr 0.15.8: <https://github.com/EmbarkStudios/cfg-expr> : (MIT License OR Apache License 2.0)
- chronotope/chrono 0.4.38: <https://github.com/chronotope/chrono> : (MIT License AND Apache License 2.0)
- clap-cargo 0.14.1 : (MIT License OR Apache License 2.0)
- clap-rs/clap 0.7.2: <https://clap.rs/> : MIT License
- clap-rs/clap 4.5.18: <https://clap.rs/> : (MIT License OR Apache License 2.0)
- clap-rs/clap 4.5.20: <https://clap.rs/> : (MIT License OR Apache License 2.0)
- colin-kiegel/rust-derive-builder 0.20.2 : Apache License 2.0
- color_quant 1.1.0: https://crates.io/crates/color_quant : MIT License
- console-rs/console 0.15.8: <https://github.com/mitsuhiko/console> : MIT License
- contain-rs/bit-set 0.5.3: <https://github.com/contain-rs/bit-set> : (MIT License OR Apache License 2.0)
- convert_case 0.6.0: https://crates.io/crates/convert_case : MIT License
- cpufeatures 0.2.14 : (MIT License OR Apache License 2.0)
- crossbeam-deque 0.8.5: <https://github.com/crossbeam-rs/crossbeam/tree/master/crossbeam-deque> : (MIT License OR Apache License 2.0)
- crossbeam-epoch 0.9.18: <https://github.com/crossbeam-rs/crossbeam> : (MIT License OR Apache License 2.0)
- crossbeam-utils 0.8.20: <https://github.com/crossbeam-rs/crossbeam/tree/master/crossbeam-utils> : (MIT License OR Apache License 2.0)
- crunchy 0.2.2: <https://crates.io/crates/crunchy> : MIT License

- derive_arbitrary 1.4.1 : (MIT License OR Apache License 2.0)
- dguo/strsim-rs 0.11.1: <https://github.com/dguo/strsim-rs> : MIT License
- digest 0.10.7 : (MIT License OR Apache License 2.0)
- directories 5.0.1: <https://crates.io/crates/directories> : (MIT License OR Apache License 2.0)
- dirs 5.0.1: <https://crates.io/crates/dirs> : (MIT License OR Apache License 2.0)
- dirs-sys 0.4.1: <https://crates.io/crates/dirs-sys> : (MIT License OR Apache License 2.0)
- displaydoc 0.2.5: <https://github.com/yaahc/displaydoc> : (MIT License OR Apache License 2.0)
- djc/quinn 0.11.5 : (MIT License OR Apache License 2.0)
- djc/quinn 0.11.8 : (MIT License OR Apache License 2.0)
- doc-comment 0.3.3: <https://crates.io/crates/doc-comment> : MIT License
- dtolnay/itoa 1.0.11 : Apache License 2.0
- dtolnay/quote 1.0.37 : Apache License 2.0
- dtolnay/ryu 1.0.18 : (Apache License 2.0 OR Boost Software License 1.0)
- dyn-stack 0.10.0 : MIT License
- encode_unicode 0.3.6: https://crates.io/crates/encode_unicode : (MIT License OR Apache License 2.0)
- encoding_rs 0.8.35: https://docs.rs/encoding_rs/ : (BSD 3-clause “New” or “Revised” License AND (MIT License OR Apache License 2.0))
- enum-as-inner 0.6.1 : (MIT License OR Apache License 2.0)
- enum-map 2.7.3: <https://crates.io/crates/enum-map> : (MIT License OR Apache License 2.0)
- enum-map-derive 0.17.0: <https://crates.io/crates/enum-map-derive> : (MIT License OR Apache License 2.0)
- equivalent 1.0.1 : (MIT License OR Apache License 2.0)
- errno 0.3.9 : (MIT License OR Apache License 2.0)
- esaxx-rs 0.1.10: <https://github.com/Narsil/esaxx-rs> : Apache License 2.0
- exr 1.73.0: <https://crates.io/crates/exr> : BSD 3-clause “New” or “Revised” License
- eyre 0.6.12: <https://github.com/yaahc/eyre> : (MIT License OR Apache License 2.0)
- fallible-iterator 0.2.0: <https://crates.io/crates/fallible-iterator> : (MIT License OR Apache License 2.0)
- fancy-regex 0.13.0: <https://github.com/fancy-regex/fancy-regex> : MIT License
- fastrand 2.1.1: <https://github.com/stjepang/fastrand> : (MIT License OR Apache License 2.0)
- fdeflate: <https://github.com/image-rs/fdeflate> : (MIT License OR Apache License 2.0)
- filetime 0.2.25: <https://github.com/alexcrichton/filetime> : (MIT License OR Apache License 2.0)
- fitzgen/bumpalo 3.16.0 : Apache License 2.0
- fixedbitset 0.4.2 : (MIT License OR Apache License 2.0)
- flate2 1.0.34: <https://github.com/alexcrichton/flate2-rs> : (MIT License OR Apache License 2.0)
- foldhash 0.1.3: <https://crates.io/crates/foldhash> : zlib License
- foreign-types 0.1.1: <https://github.com/sfackler/foreign-types> : (MIT License OR Apache License 2.0)
- foreign-types 0.3.2: <https://github.com/sfackler/foreign-types> : (MIT License OR Apache License 2.0)

- `funty 2.0.0`: <https://ferrilab.github.io/ferrilab> : MIT License
- `futures-task 0.3.31`: <https://rust-lang-nursery.github.io/futures-rs> : (MIT License OR Apache License 2.0)
- `gemm 0.17.1` : MIT License
- `gemm-c32 0.17.1`: <https://crates.io/crates/gemm-c32> : MIT License
- `gemm-c64 0.17.1`: <https://crates.io/crates/gemm-c64> : MIT License
- `gemm-common 0.17.1` : MIT License
- `gemm-f16 0.17.1`: <https://crates.io/crates/gemm-f16> : MIT License
- `gemm-f32 0.17.1`: <https://crates.io/crates/gemm-f32> : MIT License
- `gemm-f64 0.17.1`: <https://crates.io/crates/gemm-f64> : MIT License
- `generic-array 0.14.7`: <https://crates.io/crates/generic-array> : MIT License
- `getrandom 0.2.15` : (MIT License OR Apache License 2.0)
- `getset 0.1.3` : MIT License
- `gif 0.13.1`: <https://github.com/image-rs/image-gif> : (MIT License OR Apache License 2.0)
- `gimli-rs 0.31.1`: <https://docs.rs/gimli/> : Apache License 2.0
- `glob 0.3.1`: <https://github.com/rust-lang/glob> : (MIT License OR Apache License 2.0)
- `hash32 0.3.1`: <https://crates.io/crates/hash32> : (MIT License OR Apache License 2.0)
- `heck 0.4.1`: <https://github.com/withoutboats/heck> : (MIT License OR Apache License 2.0)
- `heck 0.5.0`: <https://github.com/withoutboats/heck> : (MIT License OR Apache License 2.0)
- `hermitcore/libhermit-rs 0.3.9`: <https://hermitcore.org> : BSD 3-clause “New” or “Revised” License
- `hermitcore/libhermit-rs 0.4.0`: <https://hermitcore.org> : BSD 3-clause “New” or “Revised” License
- `hf-hub 0.3.2`: <https://github.com/huggingface/hf-hub> : Apache License 2.0
- `home 0.5.9` : (MIT License OR Apache License 2.0)
- `httpdate 1.0.3`: <https://crates.io/crates/httpdate> : (MIT License OR Apache License 2.0)
- `hyperium/h2 0.4.6`: <https://github.com/crates-patch> : MIT License
- `hyperium/http 1.1.0`: <https://github.com/hyperium/http> : Apache License 2.0
- `hyperium/http-body 0.1.2`: <https://github.com/hyperium/http-body> : MIT License
- `hyperium/http-body 1.0.1`: <https://github.com/hyperium/http-body> : MIT License
- `hyperium/mime 0.3.17` : MIT License
- `hyper-rustls 0.27.3`: <https://github.com/ctz/hyper-rustls> : (MIT License OR ISC License OR Apache License 2.0)
- `hyper-tls 0.6.0`: <https://crates.io/crates/hyper-tls> : (MIT License OR Apache License 2.0)
- `hyper-util`: <https://hyper.rs> : MIT License
- `iana-time-zone 0.1.61`: <https://crates.io/crates/iana-time-zone> : (MIT License OR Apache License 2.0)
- `iana-time-zone-haiku 0.1.2`: <https://crates.io/crates/iana-time-zone-haiku> : (MIT License OR Apache License 2.0)
- `icu_locid`: <https://icu4x.unicode.org> :
- `icu_normalizer`: <https://icu4x.unicode.org> :

- icu_provider: <https://icu4x.unicode.org> :
- icu_provider_macros: <https://icu4x.unicode.org> :
- idna_adapter 1.2.0: https://docs.rs/crate/idna_adapter/latest : (MIT License OR Apache License 2.0)
- image-rs/image 0.25.5: <https://github.com/image-rs/image> : (MIT License OR Apache License 2.0)
- image-webp 0.2.0: <https://github.com/image-rs/image-webp> : (MIT License OR Apache License 2.0)
- imgref 1.11.0: <https://crates.io/crates/imgref> : (Creative Commons Zero v1.0 Universal OR Apache License 2.0)
- indenter 0.3.3: <https://github.com/yaahc/indenter> : (MIT License OR Apache License 2.0)
- instant 0.1.13 : BSD 3-clause “New” or “Revised” License
- intel-mkl-src 0.8.1: <https://crates.io/crates/intel-mkl-src> :
- intel-mkl-tool 0.8.1: <https://crates.io/crates/intel-mkl-tool> : MIT License
- interpolate_name 0.2.4: https://crates.io/crates/interpolate_name : MIT License
- ipnet 2.10.1 : (MIT License OR Apache License 2.0)
- is_ci 1.2.0: https://crates.io/crates/is_ci : ISC License
- is-terminal 0.4.13 : MIT License
- japaric/heapless 0.8.0 : (MIT License OR Apache License 2.0)
- jedisctl/rust-siphash 0.3.11: <https://docs.rs/siphasher> : Apache License 2.0
- jpeg-decoder 0.3.1: <https://crates.io/crates/jpeg-decoder> : (MIT License OR Apache License 2.0)
- js-sys 0.3.72: <https://rustwasm.github.io/wasm-bindgen/> : (MIT License OR Apache License 2.0)
- Kimundi/rustc-version-rs 0.3.3 : (MIT License OR Apache License 2.0)
- KyleMayes/clang-sys 1.8.1 : Apache License 2.0
- lazy_static 1.5.0: <https://github.com/rust-lang-nursery/lazy-static.rs> : Apache License 2.0
- lebe 0.5.2: <https://crates.io/crates/lebe> : BSD 3-clause “New” or “Revised” License
- libfuzzer-sys 0.4.7: <https://crates.io/crates/libfuzzer-sys> : (University of Illinois/NCSA Open Source License OR MIT License OR Apache License 2.0)
- libm 0.2.11: <https://crates.io/crates/libm> : MIT License
- libredox 0.1.3 : MIT License
- linux-raw-sys 0.4.14 : (MIT License OR Apache License 2.0 with Exceptions OR Apache License 2.0)
- litemap: <https://crates.io/crates/litemap> :
- lock_api 0.4.12 : (MIT License OR Apache License 2.0)
- log 0.4.22 : (MIT License OR Apache License 2.0)
- loop9 0.1.5: <https://lib.rs/loop9> : MIT License
- lru 0.12.5: <https://github.com/jeromefroe/lru-rs> : MIT License
- mackwic/colored 2.1.0: <https://github.com/mackwic/colored> : Mozilla Public License 2.0
- macro_rules_attribute 0.2.0: https://crates.io/crates/macro_rules_attribute : MIT License
- macro_rules_attribute-proc_macro 0.2.0: https://crates.io/crates/macro_rules_attribute-proc_macro : MIT License
- matklad/once_cell 1.20.2 : Apache License 2.0

- maybe-rayon 0.1.1: <https://github.com/shssoichiro/maybe-rayon> : MIT License
- memmap2 0.9.5 : (MIT License OR Apache License 2.0)
- microsoft/windows-rs 0.1.0: <https://github.com/microsoft/windows-rs> : (MIT License OR Apache License 2.0)
- microsoft/windows-rs 0.1.2: <https://github.com/microsoft/windows-rs> : MIT License
- microsoft/windows-rs 0.2.0: <https://github.com/microsoft/windows-rs> : (MIT License OR Apache License 2.0)
- microsoft/windows-rs 0.52.0: <https://github.com/microsoft/windows-rs> : (MIT License OR Apache License 2.0)
- microsoft/windows-rs 0.52.6: <https://github.com/microsoft/windows-rs> : (MIT License OR Apache License 2.0)
- microsoft/windows-rs 0.57.0: <https://github.com/microsoft/windows-rs> : Apache License 2.0
- minimal-lexical 0.2.1: <https://crates.io/crates/minimal-lexical> : (MIT License OR Apache License 2.0)
- miniz_oxide 0.8.0: https://github.com/Frommi/miniz_oxide/tree/master/miniz_oxide : (zlib License OR MIT License OR Apache License 2.0)
- mitsuhiko/indicatif 0.17.8 : MIT License
- mockito 1.5.0: <https://github.com/lipanski/mockito> : MIT License
- monostate 0.1.13: <https://crates.io/crates/monostate> : (MIT License OR Apache License 2.0)
- monostate-impl 0.1.13: <https://crates.io/crates/monostate-impl> : (MIT License OR Apache License 2.0)
- mvdnes/spin-rs 0.9.8 : MIT License
- nagisa/rust_libloading 0.8.5 : ISC License
- native-tls 0.2.12: <https://crates.io/crates/native-tls> : (MIT License OR Apache License 2.0)
- new_debug_unreachable 1.0.6: https://crates.io/crates/new_debug_unreachable : MIT License
- noop_proc_macro 0.3.0: https://crates.io/crates/noop_proc_macro : MIT License
- nox/serde_urlencoded 0.7.1 : (MIT License OR Apache License 2.0)
- ntapi 0.4.1: <https://crates.io/crates/ntapi> : (MIT License OR Apache License 2.0)
- num 0.4.3: <https://github.com/rust-num/num> : (MIT License OR Apache License 2.0)
- number_prefix 0.4.0: https://crates.io/crates/number_prefix : MIT License
- num-bigint 0.4.6: <https://github.com/rust-num/num-bigint> : (MIT License OR Apache License 2.0)
- num_cpus 1.16.0: https://crates.io/crates/num_cpus : (MIT License OR Apache License 2.0)
- num-derive 0.4.2: <https://github.com/rust-num/num-derive> : (MIT License OR Apache License 2.0)
- num_enum 0.7.3: https://crates.io/crates/num_enum : (MIT License OR Apache License 2.0 OR BSD 3-clause “New” or “Revised” License)
- num_enum_derive 0.7.3: https://crates.io/crates/num_enum_derive : (MIT License OR Apache License 2.0 OR BSD 3-clause “New” or “Revised” License)
- num-integer 0.1.46: <https://github.com/rust-num/num-integer> : (MIT License OR Apache License 2.0)
- num-iter 0.1.45: <https://github.com/rust-num/num-iter> : (MIT License OR Apache License 2.0)
- num-rational 0.4.2: <https://github.com/rust-num/num-rational> : (MIT License OR Apache License 2.0)
- num-traits 0.2.19: <https://github.com/rust-num/num-traits> : (MIT License OR Apache License 2.0)
- object-rs 0.36.5: <https://crates.io/crates/object> : Apache License 2.0
- object_store 0.10.2 : (MIT License OR Apache License 2.0)

- ocipkg 0.2.9: <https://crates.io/crates/ocipkg> : (MIT License OR Apache License 2.0)
- oci-spec 0.6.7: <https://github.com/containers/oci-spec-rs> : Apache License 2.0
- oli-obk/cargo_metadata 0.18.1 : MIT License
- onig 6.4.0: <https://crates.io/crates/onig> : MIT License
- onig_sys 69.8.1: https://crates.io/crates/onig_sys : MIT License
- openssl-macros 0.1.1: <https://crates.io/crates/openssl-macros> : (MIT License OR Apache License 2.0)
- option-ext 0.2.0: <https://github.com/soc/option-ext> : Mozilla Public License 2.0
- ordian/toml_edit 0.22.22 : Apache License 2.0
- owo-colors 4.1.0 : MIT License
- paholg/typenum 1.17.0 : (MIT License OR Apache License 2.0)
- parking_lot 0.12.3 : Apache License 2.0
- parking_lot_core 0.9.10 : (MIT License OR Apache License 2.0)
- paste 1.0.15 : (MIT License OR Apache License 2.0)
- pathsearch 0.2.0: <https://crates.io/crates/pathsearch> : MIT License
- pest-parser 2.7.14: <https://github.com/pest-parser/pest> : Apache License 2.0
- petgraph 0.6.5 : (MIT License OR Apache License 2.0)
- pgrx 0.12.8: <https://github.com/tcdi/pgrx> : MIT License
- pgrx-bindgen 0.12.8: <https://github.com/pgcentralfoundation/pgrx> : MIT License
- pgrx-macros 0.12.8: <https://github.com/tcdi/pgrx> : MIT License
- pgrx-pg-config 0.12.8: <https://github.com/tcdi/pgrx> : MIT License
- pgrx-pg-sys 0.12.8: <https://github.com/tcdi/pgrx> : MIT License
- pgrx-sql-entity-graph 0.12.8: <https://github.com/tcdi/pgrx> : MIT License
- pgrx-tests 0.12.8: <https://github.com/tcdi/pgrx> : MIT License
- pin-project-lite 0.2.15: <https://github.com/taiki-e/pin-project-lite> : (MIT License OR Apache License 2.0)
- pin-utils 0.1.0: <https://crates.io/crates/pin-utils> : (MIT License OR Apache License 2.0)
- pkg-config 0.3.31 : (MIT License OR Apache License 2.0)
- png 0.17.14: <https://crates.io/crates/png> : (MIT License OR Apache License 2.0)
- portable-atomic 1.9.0 : (MIT License OR Apache License 2.0)
- ppv-lite86 0.2.20 : (MIT License OR Apache License 2.0)
- proc-macro-crate 3.2.0 : (MIT License OR Apache License 2.0)
- proc-macro-error2 2.0.1 : (MIT License OR Apache License 2.0)
- proc-macro-error-attr2 2.0.0: <https://crates.io/crates/proc-macro-error-attr2> : (MIT License OR Apache License 2.0)
- profiling 1.0.16: <https://github.com/aclysma/profiling> : (MIT License OR Apache License 2.0)
- profiling-procmacros 1.0.16: <https://github.com/aclysma/profiling> : (MIT License OR Apache License 2.- 0)
- proptest 1.5.0: <https://altsysrq.github.io/proptest-book/proptest/index.html> : (MIT License OR Apache License 2.0)

- pulp 0.18.22 : MIT License
- qoi 0.4.1: <https://github.com/aldanor/qoi-rust> : (MIT License OR Apache License 2.0)
- quinn-udp 0.5.7 : (MIT License OR Apache License 2.0)
- radium 0.7.0: <https://github.com/mystor/radium> : MIT License
- rand_chacha 0.3.1: https://crates.io/crates/rand_chacha : MIT License
- rand_core 0.6.4: https://crates.io/crates/rand_core : (MIT License OR Apache License 2.0)
- rand_distr 0.4.3: https://crates.io/crates/rand_distr : MIT License
- rand_xorshift 0.3.0: https://crates.io/crates/rand_xorshift : (MIT License OR Apache License 2.0)
- rav1e 0.3.8: <https://github.com/xiph/rav1e> : BSD 2-clause “Simplified” License
- rav1e 0.7.1: <https://github.com/xiph/rav1e> : BSD 2-clause “Simplified” License
- raw-cpuid 10.7.0: <https://github.com/gz/rust-cpuid> : MIT License
- rayon 1.10.0: <https://github.com/rayon-rs/rayon> : (MIT License OR Apache License 2.0)
- rayon 1.12.1: <https://github.com/rayon-rs/rayon> : (MIT License OR Apache License 2.0)
- rayon-cond 0.3.0: <https://crates.io/crates/rayon-cond> : (MIT License OR Apache License 2.0)
- reborrow 0.5.5: <https://crates.io/crates/reborrow> : MIT License
- redox-os 0.5.7: <http://www.redox-os.org> : MIT License
- redox_users 0.4.6 : MIT License
- regex-syntax 0.8.5: <https://github.com/rust-lang/regex> : (MIT License OR Apache License 2.0)
- rust-aho-corasick 1.1.3: <https://github.com/BurntSushi/aho-corasick> : The Unlicense
- rust-base64 0.13.1 : (MIT License OR Apache License 2.0)
- rust-base64 0.22.1 : (MIT License OR Apache License 2.0)
- rustc-hash 1.1.0: <https://crates.io/crates/rustc-hash> : (MIT License OR Apache License 2.0)
- rustc-hash: <https://crates.io/crates/rustc-hash> : (MIT License OR Apache License 2.0)
- RustCrypto/hashees 0.10.6 : (MIT License OR Apache License 2.0)
- RustCrypto/hashees 0.10.8 : (MIT License OR Apache License 2.0)
- RustCrypto/MACs 0.12.1: <https://crates.io/crates/hmac> : (MIT License OR Apache License 2.0)
- RustCrypto/traits 0.1.6: <https://github.com/RustCrypto/traits> : (MIT License OR Apache License 2.0)
- RustCrypto/traits 0.2.0: <https://github.com/RustCrypto/traits> : (MIT License OR Apache License 2.0)
- rust-hyper 1.5.0: <https://hyper.rs> : MIT License
- rust-iter-tools/iter-tools 0.11.0 : Apache License 2.0
- rust-iter-tools/iter-tools 0.12.1 : Apache License 2.0
- rust-iter-tools/iter-tools 0.13.0 : (MIT License OR Apache License 2.0)
- rustix 0.38.39 : (MIT License OR Apache License 2.0 with Exceptions OR Apache License 2.0)
- rust-lang/backtrace-rs 0.3.74: <https://github.com/rust-lang/backtrace-rs> : (MIT License OR Apache License 2.0)
- rust-lang/cargo 0.1.8: <https://github.com/rust-lang/cargo> : (MIT License OR Apache License 2.0)
- rust-lang/hashbrown 0.15.1 : (MIT License OR Apache License 2.0)

- rust-lang-nursery/futures-rs 0.3.31: <http://rust-lang-nursery.github.io/futures-rs> : (MIT License AND Apache License 2.0)
- rust-lang/rust-bindgen 0.70.1: <https://rust-lang.github.io/rust-bindgen/> : BSD 3-clause “New” or “Revised” License
- rust-libc 0.2.161: <https://github.com/rust-lang/libc> : (MIT License OR Apache License 2.0)
- rustls: <https://github.com/ctz/rustls> : (MIT License OR ISC License OR Apache License 2.0)
- rustls-native-certs 0.8.0: <https://github.com/ctz/rustls-native-certs> : (MIT License OR ISC License OR Apache License 2.0)
- rustls-pemfile 2.2.0: <https://github.com/rustls/pemfile> : (MIT License OR ISC License OR Apache License 2.0)
- rustls-pki-types 1.10.0: <https://github.com/rustls/pki-types> : (MIT License OR Apache License 2.0)
- rustls-webpki 0.102.8 : ISC License
- rust-mio 1.0.2: <https://github.com/carllerche/mio> : MIT License
- rust-nom 7.1.3: <https://github.com/Geal/nom> : MIT License
- rust-num/num-complex 0.4.6: <https://github.com/rust-num/num-complex> : (MIT License OR Apache License 2.0)
- rust-openssl 0.10.68 : Apache License 2.0
- rust-openssl 0.9.104 : MIT License
- rust-phf 0.11.2: <https://github.com/sfackler/rust-phf> : MIT License
- rust-postgres 0.19.9: <https://github.com/sfackler/rust-postgres> : (MIT License OR Apache License 2.0)
- rust-postgres 0.2.8: <https://github.com/sfackler/rust-postgres> : (MIT License OR Apache License 2.0)
- rust-postgres 0.6.7: <https://github.com/sfackler/rust-postgres> : (MIT License OR Apache License 2.0)
- rust-postgres 0.7.12: <https://github.com/sfackler/rust-postgres> : (MIT License OR Apache License 2.0)
- rust-random/rand 0.8.5: <https://crates.io/crates/rand> : (MIT License OR Apache License 2.0)
- rust-regex 1.11.1: <https://docs.rs/regex> : (MIT License OR Apache License 2.0)
- rust-rgb 0.8.50: <https://kornelski.github.io/rust-rgb/rgb/> : MIT License
- rust-url 1.2.1: <https://docs.rs/url/> : (MIT License OR Apache License 2.0)
- rust-url 2.3.1: <https://docs.rs/url/> : Apache License 2.0
- rust-url: <https://docs.rs/url/> : (MIT License AND Apache License 2.0)
- rustversion 1.0.18 : (MIT License OR Apache License 2.0)
- rust-xattr 1.3.1: <https://github.com/Stebalien/xattr> : Apache License 2.0
- rusty-fork 0.3.0: <https://crates.io/crates/rusty-fork> : (MIT License OR Apache License 2.0)
- safetensors 0.4.5: <https://github.com/huggingface/safetensors> : Apache License 2.0
- seahash 4.1.0: <https://crates.io/crates/seahash> : MIT License
- seanmonstar/httparse 1.9.5: <https://crates.io/crates/httparse> : (MIT License OR Apache License 2.0)
- seanmonstar/request : (MIT License OR Apache License 2.0)
- seanmonstar/try-lock 0.2.5: <https://github.com/seanmonstar/try-lock> : MIT License
- seanmonstar/want 0.3.1: <https://github.com/seanmonstar/want> : MIT License

- security-framework 2.11.1 : (MIT License OR Apache License 2.0)
- security-framework 2.12.0 : (MIT License OR Apache License 2.0)
- semver 0.11.0: <https://docs.rs/crate/semver/> : (MIT License OR Apache License 2.0)
- semver 1.0.23: <https://docs.rs/crate/semver/> : (MIT License OR Apache License 2.0)
- semver-parser 0.10.2: <https://github.com/steveklabnik/semver-parser> : (MIT License OR Apache License 2.0)
- seq-macro 0.3.5: <https://crates.io/crates/seq-macro> : (MIT License OR Apache License 2.0)
- serde: <https://serde.rs/> : (MIT License OR Apache License 2.0)
- serde_cbor 0.11.2: https://crates.io/crates/serde_cbor : (MIT License OR Apache License 2.0)
- serde_json 1.0.132: http://docs.serde.rs/serde_json/ : Apache License 2.0
- serde_plain 1.0.2: https://docs.rs/serde_plain : (MIT License OR Apache License 2.0)
- serde_spanned 0.6.8: <https://github.com/toml-rs/toml> : (MIT License OR Apache License 2.0)
- SergioBenitez/version_check 0.9.5 : (MIT License OR Apache License 2.0)
- servo/core-foundation-rs 0.8.7: <https://github.com/servo/core-foundation-rs> : (MIT License OR Apache License 2.0)
- servo/core-foundation-rs 0.9.4: <https://github.com/servo/core-foundation-rs> : (MIT License OR Apache License 2.0)
- servo/rust-fnv 1.0.7 : (MIT License OR Apache License 2.0)
- servo/rust-smallvec 1.13.2 : Apache License 2.0
- servo/unicode-bidi 0.3.17 : (MIT License OR Apache License 2.0)
- shlex 1.3.0: <https://crates.io/crates/shlex> : (MIT License OR Apache License 2.0)
- signal-hook-registry 1.4.2: <https://crates.io/crates/signal-hook-registry> : (MIT License OR Apache License 2.0)
- simd-adler32 0.3.7: <https://crates.io/crates/simd-adler32> : MIT License
- simd_helpers 0.1.0: https://crates.io/crates/simd_helpers : MIT License
- similar 2.6.0 : Apache License 2.0
- slab 0.4.9: <https://github.com/carllerche/slab> : MIT License
- snafu 0.7.5: <https://crates.io/crates/snafu> : (MIT License OR Apache License 2.0)
- snafu-derive 0.7.5: <https://crates.io/crates/snafu-derive> : (MIT License OR Apache License 2.0)
- spm_precompiled 0.1.4: https://github.com/huggingface/spm_precompiled : Apache License 2.0
- sptr 0.3.2: <https://crates.io/crates/sptr> : (MIT License OR Apache License 2.0)
- srijs/rust-crc32fast 1.4.2 : Apache License 2.0
- stable_deref_trait 1.2.0: https://crates.io/crates/stable_deref_trait : (MIT License OR Apache License 2.0)
- starkat99/half-rs 1.8.3 : (MIT License OR Apache License 2.0)
- starkat99/half-rs 2.4.1 : (MIT License OR Apache License 2.0)
- Stebalien/tempfile 3.13.0: <http://stebalien.com/projects/tempfile-rs> : Apache License 2.0
- steffengy/schannel-rs 0.1.26 : MIT License
- stringprep 0.1.5: <https://crates.io/crates/stringprep> : (MIT License OR Apache License 2.0)
- strum-rs 0.26.3: <https://crates.io/crates/strum> : MIT License

- strum-rs 0.26.4: <https://crates.io/crates/strum> : MIT License
- subtle 2.6.1: <https://crates.io/crates/subtle> : BSD 3-clause “New” or “Revised” License
- supports-color 2.1.0: <https://crates.io/crates/supports-color> : Apache License 2.0
- supports-color 3.0.1: <https://crates.io/crates/supports-color> : Apache License 2.0
- syn 1.0.109: <https://github.com/dtolnay/syn> : Apache License 2.0
- syn 2.0.87: <https://github.com/dtolnay/syn> : (MIT License OR Apache License 2.0)
- sync_wrapper 1.0.1: https://docs.rs/sync_wrapper : Apache License 2.0
- synstructure 0.13.1: <https://crates.io/crates/synstructure> : MIT License
- sysctl 0.5.5 : MIT License
- sysinfo-rs 0.30.13: <https://github.com/GuillaumeGomez/sysinfo> : MIT License
- sysinfo-rs 0.32.0: <https://github.com/GuillaumeGomez/sysinfo> : MIT License
- system-configuration 0.6.1 : (MIT License OR Apache License 2.0)
- system-configuration-sys 0.6.0: <https://crates.io/crates/system-configuration-sys> : (MIT License OR Apache License 2.0)
- system-deps 6.2.2: <https://crates.io/crates/system-deps> : (MIT License OR Apache License 2.0)
- tafia/quick-xml 0.36.2 : MIT License
- tailhook/humantime 2.1.0: <https://github.com/tailhook/humantime> : (MIT License OR Apache License 2.0)
- tailhook/quick-error 1.2.3: <http://github.com/rossmacarthur/quick-error2> : (MIT License OR Apache License 2.0)
- tailhook/quick-error 2.0.1: <http://github.com/rossmacarthur/quick-error2> : (MIT License OR Apache License 2.0)
- tap 1.0.1: <https://github.com/myrilyn/tap> : MIT License
- target-lexicon 0.12.16: <https://crates.io/crates/target-lexicon> : Apache License 2.0 with Exceptions
- tar (Rust) 0.4.43: <https://github.com/alexerichon/tar-rs> : (MIT License OR Apache License 2.0)
- TedDriggs/darling 0.20.10 : MIT License
- TedDriggs/ident_case 1.0.1: https://github.com/TedDriggs/ident_case.git : (MIT License OR Apache License 2.0)
- thiserror 1.0.68 : (MIT License OR Apache License 2.0)
- thiserror-impl 1.0.68 : (MIT License OR Apache License 2.0)
- tiff 0.9.1: <https://crates.io/crates/tiff> : MIT License
- tinystr: <https://crates.io/crates/tinystr> : (MIT License OR Apache License 2.0)
- tinyvec 1.8.0: <https://crates.io/crates/tinyvec> : (zlib License OR MIT License OR Apache License 2.0)
- tinyvec_macros 0.1.1: https://crates.io/crates/tinyvec_macros : (zlib License OR MIT License OR Apache License 2.0)
- tokenizers 0.20.3: <https://github.com/epwalsh/tokenizers> : Apache License 2.0
- tokio 1.41.0: <https://tokio.rs> : MIT License
- tokio-macros 2.4.0: <https://tokio.rs> : MIT License
- tokio-native-tls 0.3.1: <https://crates.io/crates/tokio-native-tls> : MIT License

- tokio-rs/bytes 1.8.0: <https://github.com/carllerche/bytes> : MIT License
- tokio-rustls 0.26.0: <https://github.com/tokio-rs/tls> : (MIT License OR Apache License 2.0)
- tokio-util 0.7.12: <https://tokio.rs> : MIT License
- toml_datetime 0.6.8: <https://github.com/toml-rs/toml> : (MIT License OR Apache License 2.0)
- toml-rs 0.8.19: <https://crates.io/crates/toml> : (MIT License OR Apache License 2.0)
- tower-rs/tower 0.3.3: <https://github.com/tower-rs/tower> : MIT License
- tracing 0.1.40: <https://tokio.rs> : MIT License
- tracing-attributes 0.1.27: <https://tokio.rs> : MIT License
- tracing-core 0.1.32: <https://tokio.rs> : MIT License
- ug 0.0.2: <https://crates.io/crates/ug> : (MIT License OR Apache License 2.0)
- unarray 0.1.4: <https://crates.io/crates/unarray> : (MIT License OR Apache License 2.0)
- unescape 0.1.0: <https://crates.io/crates/unescape> : MIT License
- unicode_categories 0.1.1: https://crates.io/crates/unicode_categories : (MIT License OR Apache License 2.0)
- unicode-ident 1.0.13 : (Unicode License Agreement Data Files and Software (2016) AND (MIT License OR Apache License 2.0))
- unicode-normalization-alignments 0.1.12: <https://github.com/n1t0/unicode-normalization> : (MIT License OR Apache License 2.0)
- unicode-org/icu4x 0.1.4: <https://github.com/unicode-org/icu4x#readme> : Unicode License Agreement - Data Files and Software (2016)
- unicode-org/icu4x 1.5.1: <https://github.com/unicode-org/icu4x#readme> : Unicode License v3
- unicode-org/icu4x: <https://github.com/unicode-org/icu4x#readme> : Unicode License v3
- unicode-properties 0.1.3: <https://github.com/unicode-rs/unicode-properties> : (MIT License OR Apache License 2.0)
- unicode-rs/unicode-normalization 0.1.24: <https://github.com/unicode-rs/unicode-normalization> : (MIT License AND Apache License 2.0)
- unicode-rs/unicode-segmentation 1.12.0: <https://github.com/unicode-rs/unicode-segmentation> : (MIT License OR Apache License 2.0)
- unicode-rs/unicode-width 0.1.14: <https://github.com/unicode-rs/unicode-width> : (MIT License OR Apache License 2.0)
- ureq 2.10.1 : (MIT License OR Apache License 2.0)
- utf16_iter 1.0.5: https://docs.rs/utf16_iter/ : (MIT License OR Apache License 2.0)
- utf8_iter 1.0.4: https://docs.rs/utf8_iter/ : (MIT License OR Apache License 2.0)
- uuid-rs/uuid 1.11.0: <http://docs.rs/uuid> : (MIT License OR Apache License 2.0)
- vcpkg-rs 0.2.15: <https://docs.rs/vcpkg> : (MIT License OR Apache License 2.0)
- version-compare 0.2.0: <https://github.com/timvisee/version-compare> : MIT License
- wait-timeout 0.2.0: <https://github.com/alexcrichon/wait-timeout> : (MIT License OR Apache License 2.0)
- wasi 0.11.0+wasi-snapshot-preview1 : (MIT License OR Apache License 2.0 with Exceptions OR Apache License 2.0)

- wasite 0.1.0: <https://github.com/ardaku/wasite/blob/stable/CHANGELOG.md> : (MIT License OR Apache License 2.0 OR Boost Software License 1.0)
- wasm-bindgen 0.2.95: <https://github.com/rustwasm/wasm-bindgen> : (MIT License OR Apache License 2.0)
- wasm-bindgen 0.3.72: <https://github.com/rustwasm/wasm-bindgen> : (MIT License OR Apache License 2.0)
- wasm-bindgen 0.4.45: <https://github.com/rustwasm/wasm-bindgen> : (MIT License OR Apache License 2.0)
- wasm-bindgen-macro 0.2.95: <https://rustwasm.github.io/wasm-bindgen/> : (MIT License OR Apache License 2.0)
- wasm-bindgen-macro-support 0.2.95: <https://rustwasm.github.io/wasm-bindgen/> : (MIT License OR Apache License 2.0)
- wasm-bindgen-shared 0.2.95: <https://rustwasm.github.io/wasm-bindgen/> : (MIT License OR Apache License 2.0)
- wasm-streams 0.4.2 : (MIT License OR Apache License 2.0)
- webpki-roots 0.26.6: <https://github.com/ctz/webpki-roots> : Mozilla Public License 2.0
- weezl 0.1.8: <https://crates.io/crates/weezl> : (MIT License OR Apache License 2.0)
- whoami 1.5.2 : (MIT License OR Apache License 2.0 OR Boost Software License 1.0)
- winapi-i686-pc-windows-gnu 0.4.0: <https://crates.io/crates/winapi-i686-pc-windows-gnu> : (MIT License OR Apache License 2.0)
- winapi-rs 0.3.9: <https://crates.io/crates/winapi> : (MIT License AND Apache License 2.0)
- winapi-util 0.1.9: <https://github.com/BurntSushi/winapi-util> : (MIT License OR The Unlicense)
- winapi-x86_64-pc-windows-gnu 0.4.0: https://crates.io/crates/winapi-x86_64-pc-windows-gnu : (MIT License OR Apache License 2.0)
- windows_aarch64_gnullvm 0.48.5 : (MIT License OR Apache License 2.0)
- windows_aarch64_gnullvm 0.52.6 : (MIT License OR Apache License 2.0)
- windows_aarch64_msvc 0.48.5 : (MIT License OR Apache License 2.0)
- windows_aarch64_msvc 0.52.6 : (MIT License OR Apache License 2.0)
- windows-core 0.52.0 : (MIT License OR Apache License 2.0)
- windows-core 0.57.0 : (MIT License OR Apache License 2.0)
- windows_i686_gnu 0.48.5 : (MIT License OR Apache License 2.0)
- windows_i686_gnu 0.52.6 : (MIT License OR Apache License 2.0)
- windows_i686_msvc 0.48.5 : (MIT License OR Apache License 2.0)
- windows_i686_msvc 0.52.6 : (MIT License OR Apache License 2.0)
- windows-implement 0.57.0: <https://crates.io/crates/windows-implement> : (MIT License OR Apache License 2.0)
- windows-interface 0.57.0: <https://crates.io/crates/windows-interface> : (MIT License OR Apache License 2.0)
- windows-sys 0.48.0 : (MIT License OR Apache License 2.0)
- windows-sys 0.52.0 : (MIT License OR Apache License 2.0)
- windows-sys 0.59.0 : (MIT License OR Apache License 2.0)
- windows-targets 0.48.5 : (MIT License OR Apache License 2.0)

- windows-targets 0.52.6 : (MIT License OR Apache License 2.0)
- windows_x86_64_gnu 0.48.5 : (MIT License OR Apache License 2.0)
- windows_x86_64_gnu 0.52.6 : (MIT License OR Apache License 2.0)
- windows_x86_64_gnullvm 0.48.5 : (MIT License OR Apache License 2.0)
- windows_x86_64_gnullvm 0.52.6 : (MIT License OR Apache License 2.0)
- windows_x86_64_msvc 0.48.5 : (MIT License OR Apache License 2.0)
- windows_x86_64_msvc 0.52.6 : (MIT License OR Apache License 2.0)
- winnow 0.6.20 : MIT License
- write16 1.0.0: <https://docs.rs/write16/> : (MIT License OR Apache License 2.0)
- writeable: <https://crates.io/crates/writeable> :
- wyz 0.5.1: <https://myrrlyn.net/crates/wyz> : MIT License
- yansi-term 0.1.2: <https://github.com/botika/yansi-term> : MIT License
- yoke 0.7.4: <https://crates.io/crates/yoke> : Unicode License v3
- yoke-derive 0.7.4: <https://crates.io/crates/yoke-derive> : Unicode License v3
- zerocopy 0.7.35 : (MIT License OR BSD 2-clause “Simplified” License OR Apache License 2.0)
- zerocopy-derive 0.7.35 : (MIT License OR BSD 2-clause “Simplified” License OR Apache License 2.0)
- zeroize 1.8.1: <https://github.com/iqlusioninc/crates/> : (MIT License OR Apache License 2.0)
- zerovec: <https://crates.io/crates/zerovec> :
- zerovec-derive 0.10.3: <https://crates.io/crates/zerovec-derive> : Unicode License v3
- zip 1.1.4: <https://crates.io/crates/zip> : MIT License
- zune-core 0.4.12: <https://github.com/etemesi254/zune-image/tree/dev/zune-core> : (zlib License OR MIT License OR Apache License 2.0)
- zune-inflate 0.2.54: <https://github.com/etemesi254/zune-image/tree/main/zune-inflate> : (zlib License OR MIT License OR Apache License 2.0)
- zune-jpeg 0.4.13: <https://crates.io/crates/zune-jpeg> : (zlib License OR MIT License OR Apache License 2.0)

3.6 Vector Engine (PGvector)

3.6.1 Vector Engine

Vector Engine is part of **EDB Postgres® AI** (EDB PG AI), enabling advanced **vector search** and **hybrid search** capabilities directly in your Postgres databases.

It leverages the open source **PGvector** extension to provide efficient storage and similarity search for high-dimensional vector data — a foundational capability for **Retrieval-Augmented Generation (RAG)**, **semantic search**, and intelligent applications.

PGvector is an independent extension, maintained by the open source community. EDB PG AI leverages PGvector as part of its stack:

- Pipelines and data preparation can automatically populate vector columns using embedding models.
- *Gen AI Builder* and Knowledge Bases use PGvector-backed retrievals.
- You can also use PGvector independently — it is not a proprietary component.

3.6.2 Why Vector Search in Postgres?

Traditional databases excel at structured data and transactional workloads. AI-powered applications increasingly rely on **vector embeddings** — numeric representations of unstructured content such as text, images, or audio.

Vector Engine enables you to:

- Store vector embeddings in your **Postgres clusters** — no separate vector database required
- Search and retrieve content by similarity, using **Approximate Nearest Neighbor (ANN)** indexes
- Combine **vector search** + **SQL search** for hybrid queries across structured and unstructured data
- Power **Sovereign Gen AI** — keeping embeddings, metadata, and results in your Postgres estate
- Audit and govern AI search and retrieval as part of your Postgres data platform

3.6.3 Who Should Use Vector Engine?

Vector Engine is designed for:

- **Data teams** building RAG pipelines and semantic search on top of Postgres
- **Developers** building Gen AI assistants that integrate with existing Postgres data
- **Analytics teams** augmenting BI and analytics with vector-based search patterns
- **Architects** designing **Sovereign AI** solutions that must govern embeddings and results

If your application uses **vector embeddings** and you want to keep both the vectors and the source data in Postgres — Vector Engine provides a scalable, production-ready path.

3.6.4 What Does It Provide?

At its core, Vector Engine — using PGvector — adds:

- The `vector` column type — to store high-dimensional vector data
- Distance functions:
 - `vector <-> vector` — Euclidean distance (L2)
 - `vector <#> vector` — inner product
 - `vector <=> vector` — cosine distance
- Support for **Approximate Nearest Neighbor (ANN)** indexes:
 - IVFFlat index type
 - Tunable `lists` and `probes` for balancing accuracy vs. speed

PGvector itself is open source — EDB PG AI uses it to power:

- Retrieval in Knowledge Bases (via *Gen AI Builder*)
- Vector search in Pipelines-based RAG pipelines
- Hybrid AI + SQL query patterns in Postgres

3.6.5 How Vector Engine Works

1. Install and enable PGvector in your Postgres database:

```
CREATE EXTENSION IF NOT EXISTS vector;
```

1. Define vector columns in your tables:

```
ALTER TABLE documents ADD COLUMN embedding vector(768);
```

1. Index vector columns for fast similarity search:

```
CREATE INDEX ON documents USING ivfflat (embedding vector_l2_ops) WITH (lists = 100);
```

Use Pipelines and data preparation to generate embeddings

Or use any external embedding model and insert vectors manually

1. Query using similarity search:

```
SELECT id, content
FROM documents
ORDER BY embedding <-> [0.12, 0.34, 0.56, ...]::vector
LIMIT 5;
```

3.6.6 When Should You Use Vector Engine?

Use Vector Engine when:

- You are building Gen AI or AI-powered applications that require grounding in your own data
- You want to perform semantic search across documents, web pages, knowledge bases
- You need a Sovereign AI architecture — embeddings and source data fully inside your Postgres clusters
- You want to support hybrid structured + unstructured search in a unified data platform
- You are developing RAG pipelines using EDB PG AI and AI Accelerator Pipelines

3.6.7 Example Use Cases

- Enterprise RAG Assistants — combine knowledge base retrieval with LLMs
- Internal semantic search — search corporate content across multiple repositories
- Recommendation systems — similarity search for content or products
- Vision/audio similarity — image or audio embedding search

3.6.8 Integration with AI Accelerator Pipelines and Gen AI Builder

AI Accelerator [Pipelines](#) makes it easy to:

AI Factory [Pipelines](#) makes it easy to:

- Vectorize documents and populate Postgres vector columns
- Automate and orchestrate RAG pipelines
- Use Supported Embedding Models for your vectorization
- [Pipelines](#) uses PGvector-based retrieval when building Knowledge Bases:

Use Supported Embedding Models for your vectorization

Pipelines uses PGvector-based retrieval when building Knowledge Bases:

Knowledge Bases store embeddings in Postgres vector columns

RAG flows query using PGvector functions for similarity search

3.6.9 Next Steps

- Learn about AI Accelerator Pipelines
- Explore Pipeline Preparer Examples for vectorization patterns
- Review Supported Embedding Models
- Explore Gen AI Builder Knowledge Bases and RAG pipelines
- See full PGvector Documentation

EDB PG AI Vector Engine helps you build powerful Sovereign AI solutions — with your data, your embeddings, your control — directly in your Postgres estate.

3.7 AI Factory Release Notes

3.7.1 AI Factory 1.3 release notes

Release Date: June 17, 2025

We are thrilled to announce the first public availability release of **EDB Postgres AI Factory v1.3**.

Release Highlights

- **Sovereign AI & Analytics Platform:** AI Factory is now a complete platform for building modern data and AI applications. Deploy a Postgres-native lakehouse and sovereign GenAI workloads with unified governance, all within your own infrastructure.
- **New Deployment Option: Sovereign Data and AI Factory:** Introducing a fully integrated hardware and software appliance, co-designed by EDB and Supermicro®, for deploying production-grade Postgres and AI workloads in your own data center with no external dependencies.
- **Postgres-Native Lakehouse:** A groundbreaking analytics stack is now built-in. Deploy **Lakehouse Clusters** to run high-speed, vectorized queries directly against open data formats like **Apache Iceberg®** and the **Delta Lake format** in your object storage.
- **AI Factory for GenAI Workloads:** Build and serve powerful AI applications with a low-code **GenAI Builder**, integrated model serving with GPU acceleration, in-database **Semantic Search**, and a comprehensive Python SDK.
- **Unified Management & Proactive Diagnostics:** Experience a completely redesigned monitoring interface, a new cluster-level scorecard, **proactive query recommendations**, and deep query latency statistics.

Deployment Options

EDB Postgres AI Factory v1.3 is available in two ways, providing maximum flexibility for your operational needs.

Sovereign Data and AI Factory (Hardware Appliance)

For organizations requiring maximum control and zero external dependencies, the **Sovereign Data and AI Factory** is a fully integrated, pre-assembled hardware and software platform. Delivered as a rack-mounted system to your data center, it is the ideal choice for environments where data sovereignty, security, and lifecycle reliability are paramount.

This engineered system is optimized for scale and is ready to run advanced Postgres, vector, and AI pipelines the moment it's installed. It includes validated Supermicro servers, lifecycle-managed Kubernetes, and the full AI Factory software suite with optional GPU nodes for AI inference.

Software-Only Deployment

For organizations with existing infrastructure, EDB Postgres AI Factory can be deployed as a software-only solution on your preferred Kubernetes distribution. It is certified for **Amazon EKS**, **Google GKE**, and **Red Hat OpenShift (RHOS)**.

New Features & Enhancements

Sovereign AI & Analytics Platform

AI Factory v1.3 introduces a powerful, integrated suite of tools for building and managing next-generation analytics and AI workloads directly on your Postgres platform.

Postgres-Native Analytics

Build a modern, cost-effective data architecture that combines the power of Postgres with the scale of object storage. AI Factory provides the tools to manage your entire analytics stack.

- **Lakehouse Clusters:** Provision and manage dedicated compute clusters that provide scalable, vectorized query execution on object storage. They serve as the query engine for your Postgres-native lakehouse.
- **Apache Iceberg® Support:** Natively manage large analytical tables in Iceberg format. Use **Tiered Tables** with EDB Postgres Distributed (PGD) to automatically offload cold data partitions to Iceberg, making them queryable via Lakehouse Clusters while reducing storage costs.
- **Delta Lake Format Integration:** Seamlessly integrate with existing data lakes. Lakehouse Clusters can directly query tables using the Delta Lake open format, enabling fast analytics on data produced by other big data pipelines like Spark.
- **Tiered Tables with PGD:** Efficiently manage large time-series datasets by automatically partitioning tables and offloading cold partitions to object storage in Iceberg format. A single query can span both hot transactional data in Postgres and cold historical data in your data lake.

GenAI Workloads with AI Factory

EDB Postgres AI Factory v1.3 introduces the **AI Factory**, a comprehensive GenAI capability stack designed for enterprise-grade deployments.

- **GenAI Builder:** A web-based low-code interface to rapidly prototype, iterate, and deploy AI applications—from simple one-shot prompts to complex RAG pipelines.
- **Model Serving:** Deploy and manage **NVIDIA NIM** containers for AI inferencing on GPU-accelerated infrastructure, with built-in support for availability and autoscaling.

- **Semantic Search:** Use vector search capabilities built into Postgres to power retrieval for your GenAI applications. Ingest and embed text, HTML, PDFs, or images—then query them via similarity search.
- **Python SDK for AI Development:** The EDB AI Builder Framework SDK enables developers to assemble AI workflows using Python. Integrate retrieval, reranking, embeddings, and generative responses with fewer lines of code.

Observability, Monitoring & Insights

v1.3 enhances the platform's observability with expanded metrics, sustainability insights, and performance diagnostics.

- **Redesigned Monitoring Dashboard:** Gain deeper insight into cluster health with a revamped UI, improved layout, and more granular control over metrics and thresholds.
- **Query Latency Metrics:** Visualize query latency statistics collected via the `pg_stat_monitor` extension directly in the **Query Diagnostics** tab.
- **Proactive Query Recommendations:** Receive optimization tips, including new index suggestions and stale statistics alerts, based on real workload patterns.
- **Cluster Scorecard:** View a unified summary of diagnostics, recommendations, and risk indicators for each cluster in one place.
- **Self-Managed Postgres Support:** Connect external Postgres instances to AI Factory to collect query diagnostics, recommendations, and monitoring metrics.

Sustainability: ESG Reporting (Preview)

EDB AI Factory now includes **experimental support for ESG dashboards**, helping organizations track energy usage and carbon emissions for their data infrastructure.

- **Carbon and Energy Metrics:** Visualize CPU-based energy usage and estimated CO₂ emissions using configurable emission factors.
- **Region-Aware Reporting:** Customize dashboards with local energy cost (\$/kWh) and emission rate (kg CO₂/kWh) based on the deployment region.
- **Integrated Grafana Dashboards:** Access the new **Power Usage & Carbon Footprint** dashboards directly from the console.
- **Preview Phase Goals:** Validate demand for ESG observability and collect feedback from interested users.

Platform Improvements

Area	Highlights
Deployment options	New Sovereign Data and AI Factory hardware appliance.
Analytics	Lakehouse clusters with Iceberg and Delta Lake query support.
AI application tooling	GenAI Builder, model serving, semantic search, and SDK.
Monitoring	Refreshed console, scorecard view, extended query diagnostics.
Postgres coverage	Supports external/self-managed Postgres instances.
Sustainability	Preview of ESG metrics and dashboards for power and carbon tracking.

Resources

- *Installing AI Factory*
- Using AI Factory
- AI Factory Overview
- Analytics Lakehouse Architecture
- *Power & ESG Reporting*
- Release Notes Archive

Released: 30 June 2025