

AI architecture design

AI is a technology that enables machines to imitate intelligent human behavior. Machines can use AI to:

- Analyze data to create images and videos.
- Analyze and synthesize speech.
- Verbally interact in natural ways.
- Make predictions and generate new data.

You can incorporate AI into applications to perform functions or make decisions that traditional logic or processing can't handle effectively. As an architect that designs solutions, it's important to understand the AI and machine learning landscape and how you can integrate Azure solutions into your workload design.

Get started

Azure Architecture Center provides example architectures, architecture guides, architectural baselines, and ideas that you can apply to your scenario. Workloads that involve AI and machine learning components should follow the Azure Well-Architected Framework [AI workloads](#) guidance. This guidance includes principles and design guides that influence the AI and machine learning workload across the five architecture pillars. You should implement those recommendations in the scenarios and content in the Azure Architecture Center.

AI concepts

AI concepts encompass a wide range of technologies and methodologies that enable machines to perform tasks that typically require human intelligence. The following sections provide an overview of key AI concepts.

Algorithms

Algorithms or *machine learning algorithms* are pieces of code that help people explore, analyze, and find meaning in complex datasets. Each algorithm is a finite set of unambiguous step-by-step instructions that a machine can follow to achieve a specific goal. The goal of a machine learning model is to establish or discover patterns that humans can use to make predictions or categorize information. An algorithm might describe how to determine whether a pet is a cat, dog, fish, bird, or lizard. Another far more complicated algorithm might describe how to identify a written or spoken language, analyze its words, translate them into a different language, and then check the translation for accuracy.

Choose an algorithm family that best suits your task. Evaluate the various algorithms within the family to find the appropriate fit for your workload. For more information, see [What are machine learning algorithms?](#).

Machine learning

Machine learning is an AI technique that uses algorithms to create predictive models. These algorithms parse data fields and "learn" from the patterns within data to generate models. The models can then make informed predictions or decisions based on new data.

The predictive models are validated against known data, measured by performance metrics for specific business scenarios, and then adjusted as needed. This process of learning and validation is called *training*. Through periodic retraining, machine learning models improve over time.

In your workload design, you might use machine learning if your scenario includes past observations that you can reliably use to predict future situations. These observations can be universal truths, such as computer vision that detects one form of animal from another. Or these observations can be specific to your situation, such as computer vision that detects a potential assembly mistake on your assembly lines based on past warranty claim data.

For more information, see [What is machine learning?](#).

Deep learning

Deep learning is a type of machine learning that can learn through its own data processing. Like machine learning, it also uses algorithms to analyze data. But it analyzes data through artificial neural networks that contain many inputs, outputs, and layers of processing. Each layer can process the data in a different way. The output of one layer becomes the input for the next. This process enables deep learning to create more complex models than traditional machine learning.

Deep learning requires a large investment to generate highly customized or exploratory models. You might consider other solutions in this article before you add deep learning to your workload.

For more information, see [What is deep learning?](#).

Generative AI

Generative AI trains models to generate original content based on many forms of content, such as natural language, computer vision, audio, or image input. With generative AI, you can

describe a desired output in everyday language, and the model can respond by creating appropriate text, image, and code. Examples of generative AI applications include Microsoft Copilot and Azure OpenAI Service.

- [Copilot](#) is primarily a user interface that helps you write code, documents, and other text-based content. It's based on popular OpenAI models and is integrated into a wide range of Microsoft applications and user experiences.
- [Azure OpenAI](#) is a development platform as a service that provides access to OpenAI's powerful language models, such as o1-preview, o1-mini, GPT-4o, GPT-4o mini, GPT-4 Turbo with Vision, GPT-4, GPT-3.5-Turbo, and the Embeddings model series. You can adapt these models to your specific tasks, such as:
 - Content generation.
 - Content summarization.
 - Image understanding.
 - Semantic search.
 - Natural language to code translation.

Language models

Language models are a subset of generative AI that focus on natural language processing tasks, such as text generation and sentiment analysis. These models represent natural language based on the probability of words or sequences of words occurring in a given context.

Conventional language models are used in supervised settings for research purposes where the models are trained on well-labeled text datasets for specific tasks. Pretrained language models offer an accessible way to get started with AI. They're more widely used in recent years. These models are trained on large-scale text collections from the internet via deep learning neural networks. You can fine-tune them on smaller datasets for specific tasks.

The number of parameters, or weights, determine the size of a language model. Parameters influence how the model processes input data and generates output. During training, the model adjusts the weights to minimize the difference between its predictions and the actual data. This process is how the model learns parameters. The more parameters a model has, the more complex and expressive it is. But it's also more computationally expensive to train and use.

In general, small language models generally have fewer than 10 billion parameters, and large language models have more than 10 billion parameters. For example, the Microsoft Phi-3 model family has three versions:

- Mini, 3.8 billion parameters
- Small, 7 billion parameters

- Medium, 14 billion parameters

For more information, see [Language model catalog](#).

Copilots

The availability of language models led to the emergence of new ways to interact with applications and systems through digital copilots and connected, domain-specific agents. *Copilots* are generative AI assistants that integrate into applications, often as chat interfaces. They provide contextualized support for common tasks in those applications.

[Microsoft Copilot](#) integrates with a wide range of Microsoft applications and user experiences. It's based on an open architecture where non-Microsoft developers can create their own plug-ins to extend or customize the user experience with Copilot. Partner developers can also create their own copilots by using the same open architecture.

For more information, see the following resources:

- [Adopt, extend, and build Copilot experiences across the Microsoft Cloud](#)
- [Copilot Studio](#)
- [Microsoft Foundry](#)

Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) is an architecture pattern that augments the capabilities of a large language model (LLM), like ChatGPT, that's trained only on public data. You can use this pattern to add a retrieval system that provides relevant grounding data in the context with the user request. An information retrieval system provides control over grounding data that a language model uses when it formulates a response. RAG architecture helps you scope generative AI to content that's sourced from vectorized documents, images, and other data formats. RAG isn't limited to vector search storage. You can use any data store technology.

For more information, see [Design and develop a RAG solution](#) and [Choose an Azure service for vector search](#).

Agent-based architecture

For guidance about how to coordinate multiple agents in complex AI scenarios, see [AI agent orchestration patterns](#).

Azure AI services

With [Azure AI services](#), developers and organizations can use ready-made, prebuilt, and customizable APIs and models to create intelligent, market-ready, and responsible applications. Use cases include natural language processing for conversations, search, monitoring, translation, speech, vision, and decision-making.

For more information, see the following resources:

- [Choose an Azure AI services technology](#)
- [Azure AI services documentation](#)
- [Choose a natural language processing technology in Azure](#)

AI language models

LLMs, such as the OpenAI GPT models, are powerful tools that can generate natural language across various domains and tasks. To choose a model, consider factors such as data privacy, ethical use, accuracy, and bias.

[Phi open models](#) are small, less compute-intensive models for generative AI solutions. A small language model might be more efficient, interpretable, and explainable than an LLM.

When you design a workload, you can use language models as a hosted solution behind a metered API. Alternatively, for many small language models, you can host language models in-process or at least on the same compute as the consumer. When you use language models in your solution, consider your choice of language model and its available hosting options to help ensure an optimized solution for your use case.

AI development platforms and tools

The following AI development platforms and tools can help you build, deploy, and manage machine learning and AI models.

Azure Machine Learning

Azure Machine Learning is a machine learning service that you can use to build and deploy models. Machine Learning offers web interfaces and SDKs for you to train and deploy your machine learning models and pipelines at scale. Use these capabilities with open-source Python frameworks, such as PyTorch, TensorFlow, and scikit-learn.

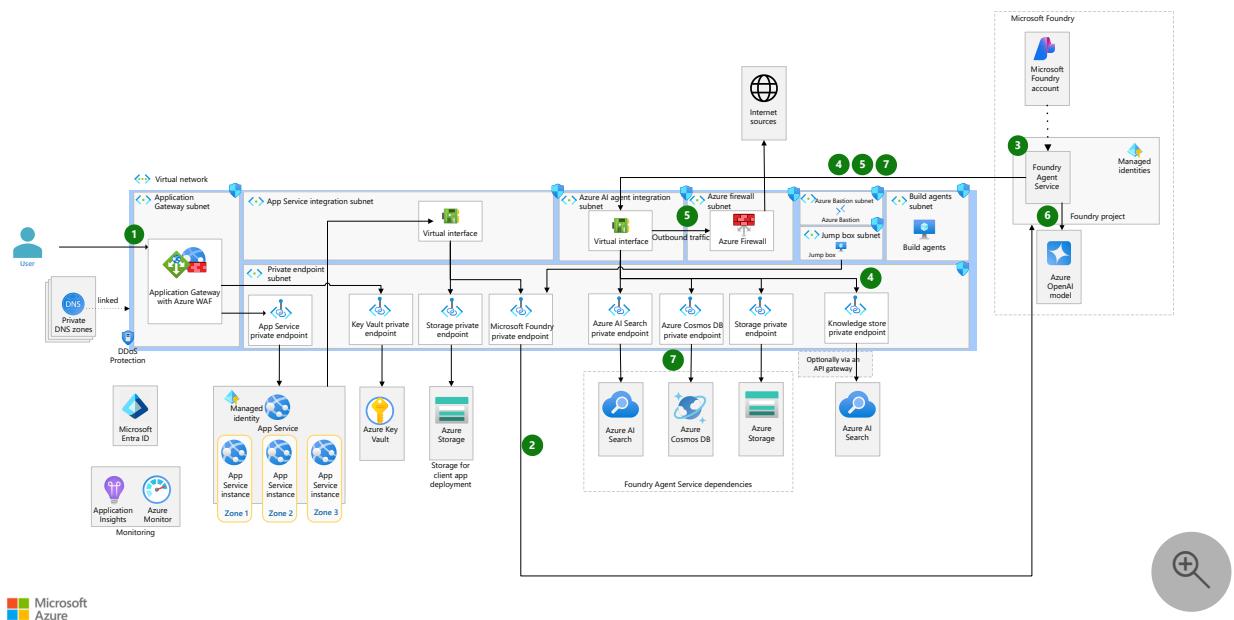
For more information, see the following resources:

- [Compare Microsoft machine learning products and technologies](#)
- [Machine Learning documentation](#)

- What is Machine Learning?

AI and Machine learning reference architectures for Azure

- Microsoft Foundry chat architecture in an Azure landing zone
- Baseline Microsoft Foundry chat reference architecture describes how to build an end-to-end chat architecture by using OpenAI's GPT models in Microsoft Foundry. It incorporates grounding via enterprise data sources to enrich responses with contextual information.



Automated machine learning

Automated machine learning (AutoML) is the process of automating the time-consuming, iterative tasks of machine learning model development. Data scientists, analysts, and developers can use AutoML to build machine learning models that have high scale, efficiency, and productivity while sustaining model quality.

For more information, see the following resources:

- What is AutoML?
- Tutorial: Train a classification model with AutoML in Machine Learning studio
- Configure AutoML experiments in Python
- Use the CLI extension for Machine Learning

MLflow

Machine Learning workspaces are MLflow-compatible, which means that you can use a Machine Learning workspace the same way that you use an MLflow server. This compatibility

provides the following advantages:

- Machine Learning doesn't host MLflow server instances but can use the MLflow APIs directly.
- You can use a Machine Learning workspace as your tracking server for any MLflow code, whether or not it runs in Machine Learning. You need to configure MLflow to point to the workspace where the tracking should occur.
- You can run training routines that use MLflow in Machine Learning without making any changes.

For more information, see [MLflow and Machine Learning](#) and [MLflow](#).

Generative AI tools

- [Microsoft Foundry](#) helps you experiment, develop, and deploy generative AI apps and APIs responsibly with a comprehensive platform. The [Microsoft Foundry portal](#) provides access to Azure AI services, foundation models, a playground, and resources to help you fine-tune, evaluate, and deploy AI models and AI agents.

[Azure AI Agent Service](#) hosts no-code agents that you define, connected to a foundation model in the AI model catalog and optionally your own custom knowledge stores or APIs. This capability is hosted within Foundry.

- [Copilot Studio](#) extends Copilot in Microsoft 365. You can use Copilot Studio to build custom copilots for internal and external scenarios. Use a comprehensive authoring canvas to design, test, and publish copilots. You can easily create generative AI-enabled conversations, provide greater control of responses for existing copilots, and accelerate productivity by using automated workflows.

Data platforms for AI

The following platforms offer comprehensive solutions for data movement, processing, ingestion, transformation, real-time analytics, and reporting.

Microsoft Fabric

Microsoft Fabric is an end-to-end analytics and data platform for enterprises that require a unified solution. You can grant workload teams access to data within Fabric. The platform covers data movement, processing, ingestion, transformation, real-time event routing, and report building. It offers a comprehensive suite of services, including Fabric Data Engineer, Fabric Data Factory, Fabric Data Science, Fabric Real-Time Intelligence, Fabric Data Warehouse, and Fabric Databases.

Fabric integrates separate components into a cohesive stack. Instead of relying on different databases or data warehouses, you can centralize data storage with OneLake. AI capabilities are embedded within Fabric, which eliminates the need for manual integration.

For more information, see the following resources:

- [What is Fabric?](#)
- [Learning path: Get started with Fabric](#)
- [AI services in Fabric](#)
- [Use Azure OpenAI in Fabric with REST API](#)
- [Use Fabric for generative AI: A guide to building and improving RAG systems ↗](#)
- [Build custom AI applications with Fabric: Implement RAG for enhanced language models ↗](#)

Copilots in Fabric

You can use Copilot and other generative AI features to transform and analyze data, generate insights, and create visualizations and reports in Fabric and Power BI. You can build your own copilot or choose one of the following prebuilt copilots:

- [Copilot in Fabric](#)
- [Copilot for Data Science and Data Engineer](#)
- [Copilot for Data Factory](#)
- [Copilot for Data Warehouse](#)
- [Copilot for Power BI](#)
- [Copilot for Real-Time Intelligence](#)

AI skills in Fabric

You can use the Fabric AI skill feature to configure a generative AI system to generate queries that answer questions about your data. After you configure an AI skill, you can share it with your colleagues, who can then ask their questions in simple language. Based on their questions, the AI generates queries on the data that answers those questions.

For more information, see the following resources:

- [What is the AI skill feature in Fabric?](#)
- [How to create an AI skill](#)
- [AI skill example](#)
- [Difference between an AI skill and a copilot](#)

Apache Spark-based data platforms for AI

Apache Spark is a parallel processing framework that supports in-memory processing to boost the performance of big data analytic applications. Spark provides basic building blocks for in-memory cluster computing. A Spark job can load and cache data into memory and query it repeatedly, which is faster than disk-based applications, such as Hadoop.

Apache Spark in Microsoft Fabric

Fabric Runtime is an Azure-integrated platform based on Apache Spark that enables the implementation and management of data engineering and data science experiences. Fabric Runtime combines key components from internal and open-source sources, which provides a comprehensive solution.

Fabric Runtime has the following key components:

- **Apache Spark** is a powerful open-source distributed computing library that enables large-scale data processing and analytics tasks. Apache Spark provides a versatile and high-performance platform for data engineering and data science experiences.
- **Delta Lake** is an open-source storage layer that integrates atomicity, consistency, isolation, and durability (ACID) transactions and other data reliability features with Apache Spark. Integrated within Fabric Runtime, Delta Lake enhances data processing capabilities and helps ensure data consistency across multiple concurrent operations.
- **Default-level packages for Java, Scala, Python, and R** are packages that support diverse programming languages and environments. These packages are automatically installed and configured, so developers can apply their preferred programming languages for data processing tasks.

Fabric Runtime is built on a robust open-source operating system to help ensure compatibility with various hardware configurations and system requirements.

For more information, see [Apache Spark runtimes in Fabric](#).

Azure Databricks Runtime for Machine Learning

[Azure Databricks](#) is an Apache Spark-based analytics platform that has one-click setup, streamlined workflows, and an interactive workspace for collaboration between data scientists, engineers, and business analysts.

You can use [Databricks Runtime for Machine Learning](#) to start a Databricks cluster with all the libraries required for distributed training. This feature provides an environment for machine learning and data science. It contains multiple popular libraries, including TensorFlow, PyTorch, Keras, and XGBoost. It also supports distributed training via Horovod.

For more information, see the following resources:

- [Azure Databricks documentation](#)
- [Machine learning capabilities in Azure Databricks](#)
- [Deep learning overview for Azure Databricks](#)

Apache Spark in Azure HDInsight

[Apache Spark in Azure HDInsight](#) is the Microsoft implementation of Apache Spark in the cloud. Spark clusters in HDInsight are compatible with Azure Storage and Azure Data Lake Storage, so you can use HDInsight Spark clusters to process data that you store in Azure.

[SynapseML](#), formerly known as MMLSpark, is the Microsoft machine learning library for Apache Spark. This open-source library adds many deep learning and data science tools, networking capabilities, and production-grade performance to the Spark ecosystem.

For more information, see the following resources:

- [SynapseML features and capabilities](#)
- [HDInsight overview](#)
- [Tutorial: Build an Apache Spark machine learning application in HDInsight](#)
- [Apache Spark best practices on HDInsight](#)
- [Configure HDInsight Apache Spark cluster settings](#)
- [Create an Apache Spark machine learning pipeline on HDInsight](#)

Data storage for AI

You can use the following platforms to efficiently store, access, and analyze large volumes of data.

Fabric OneLake

OneLake in Fabric is a unified and logical data lake that you can tailor to your entire organization. It serves as the central hub for all analytics data and is included with every Fabric tenant. OneLake in Fabric is built on the foundation of Data Lake Storage.

OneLake in Fabric:

- Supports structured and unstructured file types.
- Stores all tabular data in Delta-Parquet format.
- Provides a single data lake within tenant boundaries that's governed by default.

- Supports the creation of workspaces within a tenant so that your organization can distribute ownership and access policies.
- Supports the creation of various data items, such as lakehouses and warehouses, from which you can access data.

For more information, see [OneLake, the OneDrive for data](#).

Data Lake Storage

Data Lake Storage is a single, centralized repository where you can store your structured and unstructured data. Use a data lake to quickly and easily store, access, and analyze a wide variety of data in a single location. You don't need to conform your data to fit an existing structure. Instead, you can store your data in its raw or native format, usually as files or as binary large objects, or blobs.

Data Lake Storage provides file system semantics, file-level security, and scale. Because these capabilities are built on Azure Blob Storage, you also get low-cost, tiered storage that has high availability and disaster recovery capabilities.

Data Lake Storage uses the infrastructure of Azure Storage to create a foundation for building enterprise data lakes on Azure. Data Lake Storage can service multiple petabytes of information while sustaining hundreds of gigabits of throughput so that you can manage massive amounts of data.

For more information, see the following resources:

- [Introduction to Data Lake Storage](#)
- [Tutorial: Data Lake Storage, Azure Databricks, and Spark](#)

Data processing for AI

You can use the following tools to prepare data for machine learning and AI applications. Ensure that your data is clean and structured so that you can use it for advanced analytics.

Fabric Data Factory

You can use Fabric Data Factory to ingest, prepare, and transform data from multiple data sources, such as databases, data warehouses, lakehouses, and real-time data streams. This service can help you meet your data operations requirements when you design workloads.

Fabric Data Factory supports code solutions and no-code or low-code solutions:

- Use [data pipelines](#) to create workflow capabilities at cloud scale. Use the drag-and-drop interface to build workflows that can refresh your dataflow, move petabyte-size data, and define control-flow pipelines.
- Use [dataflows](#) as a low-code interface to ingest data from hundreds of data sources and transform it by using over 300 data transformations.

For more information, see [Data Factory end-to-end scenario: Introduction and architecture](#).

Azure Databricks

You can use the Databricks Data Intelligence Platform to write code to create a machine learning workflow by using feature engineering. *Feature engineering* is the process of transforming raw data into features that you can use to train machine learning models. Databricks Data Intelligence Platform includes key features that support feature engineering:

- **Data pipelines** ingest raw data, create feature tables, train models, and perform batch inference. When you use feature engineering in Unity Catalog to train and log a model, the model is packaged with feature metadata. When you use the model for batch scoring or online inference, it automatically retrieves feature values. The caller doesn't need to know about the values or include logic to look up or join features to score new data.
- **Model and feature serving endpoints** are instantly accessible and provide milliseconds of latency.
- **Monitoring** helps ensure the performance and accuracy of data and models.

You can also use [Mosaic AI Vector Search](#) to store and retrieve embeddings. Embeddings are crucial for applications that require similarity searches, such as RAG, recommendation systems, and image recognition.

For more information, see [Azure Databricks: Serve data for machine learning and AI](#).

Data connectors for AI

Azure Data Factory and Azure Synapse Analytics pipelines support many data stores and formats via copy, data flow, look up, get metadata, and delete activities. To see the available data store connectors, supported capabilities including the corresponding configurations, and generic Open Database Connectivity options, see [Azure Data Factory and Azure Synapse Analytics connector overview](#).

Custom AI

Custom AI solutions help you address specific business needs and challenges. The following sections provide an overview of various tools and services that you can use to build and manage custom AI models.

Azure Machine Learning

Azure Machine Learning is a cloud service for accelerating and managing the machine learning project lifecycle. Machine learning professionals, data scientists, and engineers can use this service in their day-to-day workflows to train and deploy models and manage machine learning operations.

Machine Learning offers the following capabilities:

- **Algorithm selection:** Some algorithms make specific assumptions about data structure or desired results. Choose an algorithm that fits your needs so that you can get more useful results, more accurate predictions, and faster training times. For more information, see [How to select algorithms for Machine Learning](#).
- **Hyperparameter tuning or optimization:** You can use this manual process to find hyperparameter configurations that result in the best performance. This optimization incurs significant computational costs. *Hyperparameters* are adjustable parameters that provide control in the model training process. For example, you can choose the number of hidden layers and the number of nodes in each layer of neural networks. Model performance depends heavily on hyperparameters.

You can use Machine Learning to automate hyperparameter tuning and run experiments in parallel to efficiently optimize hyperparameters.

For more information, see the following resources:

- [Hyperparameter tuning a model with Machine Learning](#)
- [Upgrade hyperparameter tuning to SDK v2](#)
- [Learning path: Perform hyperparameter tuning with Machine Learning](#)

- **Model training:** You can iteratively use an algorithm to create or *teach* models. After models are trained, you can use them to analyze data and make predictions.

During the training phase:

1. A quality set of known data is tagged so that individual fields are identifiable.
2. An algorithm that's configured to make a particular prediction receives the tagged data.

3. The algorithm outputs a model that captures the patterns that it identified in the data. The model uses a set of parameters to represent these patterns.

During validation:

1. Fresh data is tagged and used to test the model.
2. The algorithm is adjusted as needed and possibly does more training.
3. The testing phase uses real-world data without any tags or preselected targets. If the model's results are accurate, it's ready for use and can be deployed.

For more information, see the following resources:

- [Train models with Machine Learning](#)
- [Tutorial: Train a model in Machine Learning](#)
- [Deep learning and distributed training with Machine Learning](#)
- **AutoML:** This process automates the time-consuming, iterative tasks of machine learning model development. It can significantly reduce the time that it takes to produce production-ready machine learning models. AutoML can assist with model selection, hyperparameter tuning, model training, and other tasks, without requiring extensive programming or domain knowledge.

You can use AutoML when you want Machine Learning to use a specified target metric to train and tune a model. You don't need data science expertise to identify an end-to-end machine learning pipeline for problems.

Machine learning professionals and developers across industries can use AutoML to:

- Implement machine learning solutions without extensive programming or machine learning knowledge.
- Save time and resources.
- Apply data science best practices.
- Provide agile problem-solving.

For more information, see [What is AutoML?](#).

- **Scoring:** This process, also called *prediction*, uses a trained machine learning model to generate values based on new input data. The values, or scores, can represent predictions of future values, but they might also represent a likely category or outcome.

For more information, see the following resources:

- [Score model](#)
- [Deploy models for scoring in batch endpoints](#)

- **Feature engineering and featurization:** Training data consists of rows and columns. Each row is an observation or record, and the columns of each row are the features that describe each record. Typically, the features that best characterize the patterns in the data are selected to create predictive models.

Although you can use many of the raw data fields to train a model, you might need to create other engineered features that provide information to better differentiate patterns in the data. This process is called feature engineering, where you use domain knowledge of the data to create features that help machine learning algorithms learn better.

In Machine Learning, data-scaling and normalization techniques are applied to make feature engineering easier. Collectively, these techniques and feature engineering are called *featurization* in AutoML experiments. For more information, see [Data featurization in automated machine learning](#).

Azure OpenAI

In Azure OpenAI, you can use a process known as *fine-tuning* to tailor OpenAI models to your personal datasets. This customization step optimizes the service by providing:

- Higher quality results compared to [prompt engineering](#) only.
- The ability to train on more examples than a model's maximum request context limit typically permits.
- Token savings because of shorter prompts.
- Lower-latency requests, particularly when you use smaller models.

For more information, see the following resources:

- [Customize a model with fine-tuning](#)
- [Tutorial: Azure OpenAI GPT-4o-mini fine-tuning](#)
- [Baseline Microsoft Foundry chat reference architecture](#)

Azure AI services for custom AI

[Azure AI services](#) provides features to build custom AI models and applications. The following sections provide an overview of these key features.

Custom speech

[Custom speech](#) is a feature of the Azure AI Speech service. You can use custom speech to evaluate and improve the accuracy of speech recognition for your applications and products.

Use a custom speech model for real-time speech to text, speech translation, and batch transcription.

By default, speech recognition uses a universal language model as a base model. This model is trained with Microsoft-owned data and reflects commonly used spoken language. The base model is pretrained with dialects and phonetics that represent various common domains. When you make a speech recognition request, the most recent base model for your supported language is used by default. The base model works well in most speech recognition scenarios.

You can use a custom model to augment the base model. For example, you can improve the recognition of domain-specific vocabulary that's specific to an application by providing text data to train the model. You can also improve recognition for specific audio conditions of an application by providing audio data, including reference transcriptions.

If the data follows a pattern, you can use structured text to train a model. You can specify custom pronunciations and customize display text formatting with custom inverse text normalization, custom rewrite, and custom profanity filtering.

Custom translator

[Custom translator](#) is a feature of the [Azure AI Translator](#) service. Enterprises, app developers, and language service providers can use custom translator to build customized neural machine translation (NMT) systems. The customized translation systems integrate into existing applications, workflows, and websites.

You can use this feature to build and publish custom translation systems to and from English. Custom translator supports more than three dozen languages that map directly to the languages for NMT. For a complete list of languages, see [Translator language support](#).

Custom translator offers the following features.

[] [Expand table](#)

| Feature | Description |
|---|--|
| Apply NMT technology ↗ | Apply NMT from the custom translator to improve your translation. |
| Build systems that know your business terminology | Customize and build translation systems by using parallel documents that understand the terminology in your business and industry. |
| Use a dictionary to build your models | Train a model with only dictionary data if you don't have a training dataset. |
| Collaborate with others | Collaborate with your team by sharing your work with various people. |

| Feature | Description |
|--|--|
| Access your custom translation model | Access your custom translation model anytime by using your existing applications or programs via Microsoft Translator Text API V3. |

Azure AI Document Intelligence custom models

[Azure AI Document Intelligence](#) uses advanced machine learning technology to identify documents, detect and extract information from forms and documents, and return the extracted data in a structured JSON output. Use Document Intelligence to take advantage of prebuilt or pretrained document analysis models or trained standalone custom models.

[Document Intelligence custom models](#) include custom classification models for scenarios where you need to identify the document type before you invoke the extraction model. You can pair a classification model with a custom extraction model to analyze and extract fields from forms and documents that are specific to your business. Combine standalone custom extraction models to create [composed models](#).

Custom AI tools

Prebuilt AI models are useful and increasingly flexible, but the best way to optimize AI is to tailor a model to your specific needs. Two primary tools to create custom AI models are generative AI and traditional machine learning.

Azure Machine Learning studio

[Azure Machine Learning studio](#) is a cloud service for accelerating and managing the machine learning project lifecycle. Machine learning professionals, data scientists, and engineers can use it in their day-to-day workflows to train and deploy models and manage machine learning operations.

- Build and train Machine Learning models by using any type of compute, including Spark and GPUs for cloud-scale large AI workloads.
- Run AutoML and use the drag-and-drop UI for low-code Machine Learning.
- Implement end-to-end Machine Learning operations and repeatable pipelines.
- Use the responsible AI dashboard for bias detection and error analysis.
- Orchestrate and manage prompt engineering and LLM flows.
- Deploy models via REST API endpoints, real-time inference, and batch inference.

- Use hub workspaces to share compute, quota, security, and connectivity to company resources, while centralizing governance for IT. Set up a hub once, then create secure workspaces directly from the studio for each project. Use hubs to manage your team's work in the studio and the [Microsoft Foundry portal](#).

Microsoft Foundry

[Microsoft Foundry](#) helps you efficiently build and deploy custom generative AI applications with the power of broad Azure AI offerings.

- Build together as one team. Your Foundry account provides enterprise-grade security and a collaborative environment that includes shared resources and connections to pretrained models, data, and compute.
- Organize your work. Your Foundry project helps you save state so that you can iterate from the first idea to the first prototype and first production deployment. Easily invite others to collaborate with you.
- Use your preferred development platform and frameworks, including GitHub, Visual Studio Code, Microsoft Agent Framework, Semantic Kernel, and AutoGen.
- Discover and benchmark from over 1,600 models.
- Provision models as a service (MaaS) through serverless APIs and hosted fine-tuning.
- Incorporate multiple models, data sources, and modalities.
- Build RAG by using your protected enterprise data, without the need for fine-tuning.
- Orchestrate and manage prompt engineering and LLM flows.
- Design and safeguard apps and APIs via configurable filters and controls.
- Evaluate model responses by using built-in and custom evaluation flows.
- Deploy AI innovations to the Azure-managed infrastructure to provide continuous monitoring and governance across environments.
- Continuously monitor deployed apps for safety, quality, and token consumption in production.

For more information, see [Foundry portal versus Machine Learning studio](#).

Azure AI Agent Service in the Foundry portal

Azure AI Agent Service is a tool that uses to create AI agents using a no-code and nondeterministic approach. The agents are exposed as microservices on the Foundry account.

Each agent connects to a foundation model from the Azure AI model catalog. Agents can optionally connect to your own custom private knowledge stores or public data. Likewise, agents can invoke tools to perform tasks to call into custom code.

Custom AI code languages

The core concept of AI is the use of algorithms to analyze data and generate models to describe, or score, it in useful ways. Developers and data scientists, and sometimes other algorithms, use programming code to write algorithms. Two of the most popular programming languages for AI development are Python and R.

[Python](#) is a general-purpose, high-level programming language. It has a simple, easy-to-learn syntax that emphasizes readability. There's no compiling step. Python has a large standard library, and it supports the ability to add modules and packages. This feature encourages modularity and lets you expand capabilities when needed. There's a large and growing ecosystem of AI and machine learning libraries for Python, including many in Azure.

For more information, see the following resources:

- [Python on Azure product home page](#)
- [Azure for Python developers](#)
- [Machine Learning SDK for Python](#)
- [Introduction to machine learning with Python and notebooks](#)
- [scikit-learn open-source machine learning library for Python](#)
- [PyTorch open-source Python library](#)
- [TensorFlow open-source symbolic math library](#)
- [Tutorial: Apply machine learning models in Azure Functions with Python and TensorFlow](#)

[R](#) is a language and environment for statistical computing and graphics. You can use it for everything from mapping broad social and marketing trends online to developing financial and climate models.

Microsoft fully embraces the R programming language and provides many options for R developers to run their code in Azure.

For more information, see [Use R interactively on Machine Learning](#).

For general information about custom AI on Azure, see the following resources:

- [Microsoft AI on GitHub: Samples, reference architectures, and best practices](#)
- [Machine Learning SDK for Python](#)

- Machine Learning examples repository ↗
- Train R models by using the Machine Learning CLI v2 ↗

Customer stories

Many industries apply AI in innovative and inspiring ways. Consider the following customer case studies and success stories:

- Healthcare for all with Kry using Azure OpenAI ↗
- PIMCO boosts client service with an AI-powered search platform built on Azure AI ↗
- Legrand and Azure OpenAI: Powering smarter solutions with AI-driven tools ↗
- C.H. Robinson overcomes decades-old barriers to automate the logistics industry by using Azure AI ↗

[Browse more AI customer stories ↗](#)

General information about Microsoft AI

Learn more about Microsoft AI, and stay up to date with related news:

- [Microsoft AI ↗](#)
- [AI learning hub](#)
- [Azure AI ↗](#)
- [Microsoft AI news ↗](#)
- [Microsoft AI on GitHub: Samples, reference architectures, and best practices ↗](#)

Next step

- [AI workloads on Azure](#)

Related resource

- [Architecture diagrams and technology descriptions for AI solutions reference architectures](#)

Build a conversation knowledge mining solution by using Azure AI services

Azure AI services

Azure Container Apps

Azure Cosmos DB

Semantic Kernel

Azure AI Foundry

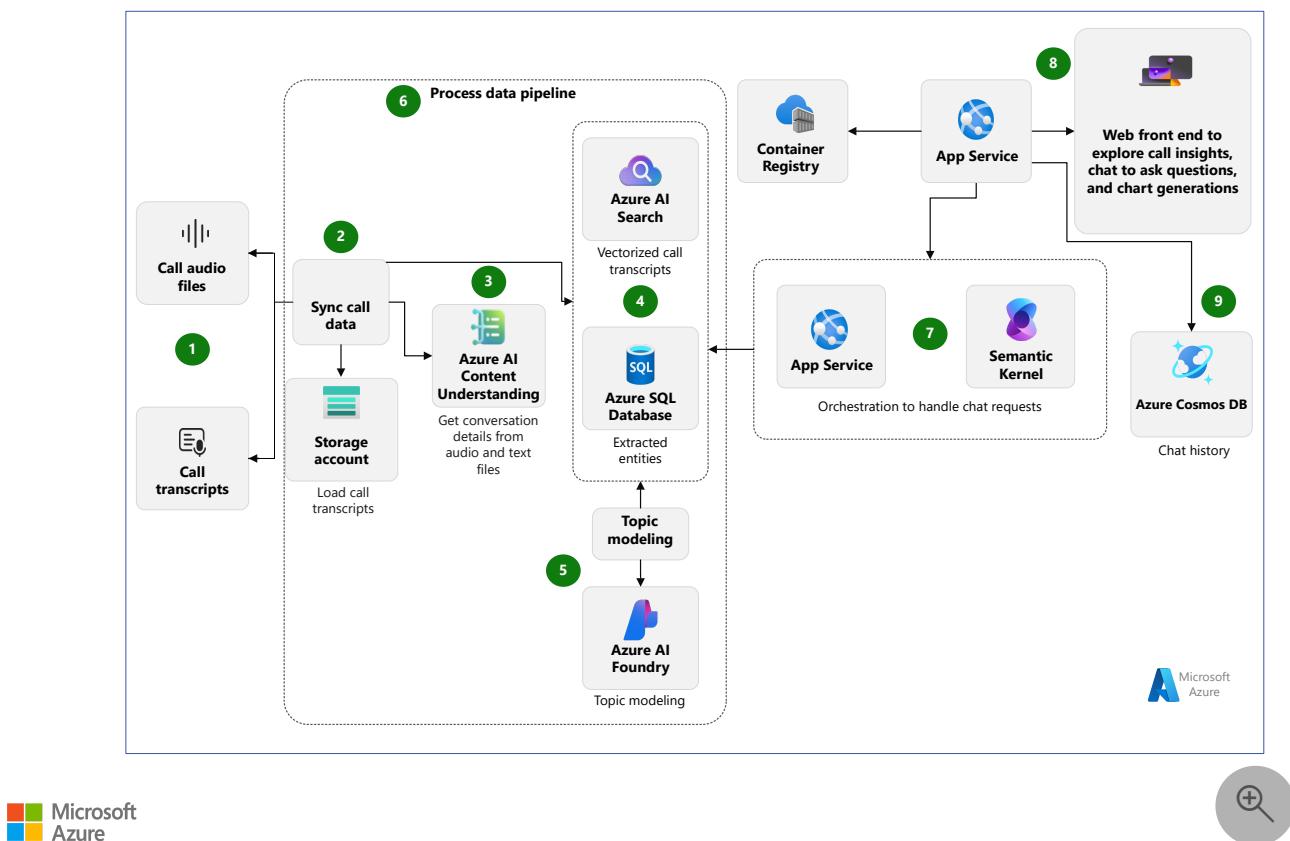
Solution ideas

This article describes a solution idea. Your cloud architect can use this guidance to help visualize the major components for a typical implementation of this architecture. Use this article as a starting point to design a well-architected solution that aligns with your workload's specific requirements.

This architecture shows a conversation knowledge mining solution that extracts actionable insights from large volumes of conversational data, such as from a call center. The solution uses Azure AI Content Understanding, Azure AI Search, Azure OpenAI, and supporting Azure services to analyze unstructured dialogue and transform it into meaningful, structured insights through natural language processing and vector-based search capabilities.

The architecture demonstrates how to build scalable conversation analytics pipelines that process audio and text inputs through event-driven workflows, extract entities and relationships, and enable interactive exploration of insights through natural language chat interfaces for enterprise-grade conversation intelligence.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

The following workflow corresponds to the preceding diagram:

1. Raw call audio files and call transcripts exist in their source systems.
2. The audio files and transcripts are uploaded to an Azure Storage account as the initial data sources. The data processing pipeline analyzes audio and transcript files from customer service calls and creates a searchable knowledge base. In this ingestion phase, audio files are converted to text by using speech-to-text technology, while transcripts are directly processed for content analysis. This process occurs daily, but your scenario might require a different frequency.
3. Content Understanding processes both audio and text files to extract conversation details, entities, relationships, and contextual information. This service performs topic modeling and key phrase extraction to identify meaningful patterns within the conversational data. This data includes call-specific elements such as resolution status, customer satisfaction indicators, and compliance markers.
4. Extracted entities and processed conversation data are stored in Azure SQL Database for structured queries, while AI Search creates vectorized representations of call transcripts.

These embeddings enable semantic search across the full set of transcripts and support complex queries related to call outcomes, agent performance, and customer sentiment trends.

5. Custom application code performs topic modeling on the extracted call transcript data from step 3. It automatically identifies and categorizes conversation themes, such as billing problems, technical support, or product feedback. This process uses Azure OpenAI models hosted in Azure AI Foundry as a managed service. Azure AI Foundry Agent Service orchestrates the workflow to analyze transcripts and extract topics. Topic modeling applies machine learning algorithms to discover patterns in language usage and group related words or phrases by assigning topic labels and confidence scores to each conversation. Results are saved to SQL Database, which enables automatic categorization and insight generation from customer calls at scale.
6. A scheduled batch processing pipeline runs periodically to process new data stored in the Storage account from steps 1 and 2. The pipeline uses Content Understanding services to analyze call audio files and transcripts. It extracts insights and transforms raw data into structured, searchable information. The processed results are written to AI Search, which stores vectorized call transcripts to enable semantic search capabilities, and SQL Database, which stores extracted entities and structured data. The pipeline can operate in single-threaded mode for sequential processing. Or it can operate in parallel across multiple threads to handle larger data volumes and improve processing throughput, depending on workload requirements and system capacity.
7. The orchestration layer within Azure App Service coordinates the overall workflow by managing data flow between services and providing API endpoints. It integrates Azure OpenAI models in Azure AI Foundry and Semantic Kernel for intelligent processing and response generation by using function calling capabilities to enhance the conversation analysis workflow. This orchestration is only for handling chat requests.
8. Users access a web front end hosted on App Service to explore call insights, chat with the data by using natural language queries, and generate visualizations. The interface provides conversational access to the processed knowledge base and enables queries such as *Show me all unresolved billing complaints from last month* or *What are the most common reasons for escalations?*
9. Azure Cosmos DB stores chat history and session data. It maintains conversation context for the interactive front-end experience and enables persistent user sessions across the application. The chat history is stored in Azure Cosmos DB, so Azure Cosmos DB only interacts with the app to pull the user's previous questions and answers. The data to be queried for new questions is in SQL Database and the AI Search index.

Components

- A [Storage account](#) is a scalable cloud storage service that provides secure and durable storage for various data types. In this architecture, the Storage account serves as the primary ingestion point for call audio files and transcripts. It provides a reliable foundation for the conversation analysis pipeline and supports hot, cool, and archive storage tiers to optimize costs for long-term conversation data retention.
- [Content Understanding](#) is an AI service that extracts insights from unstructured content, including audio and text. In this architecture, Content Understanding processes conversational data to identify entities, relationships, and key themes. It transforms raw dialogue into structured, analyzable information, including call center-specific insights such as resolution indicators, escalation triggers, and compliance markers.
- [AI Search](#) is a cloud search service that provides rich search capabilities over user-generated content. In this architecture, AI Search creates and manages vectorized representations of call transcripts. This approach enables semantic search and retrieval-augmented generation (RAG) patterns for intelligent conversation exploration. It uses optimized indexing strategies to handle millions of conversation records efficiently.
- [SQL Database](#) is a fully managed relational database service that provides high availability and scalability. In this architecture, SQL Database stores extracted entities, conversation metadata, and structured insights. This configuration enables efficient querying and analysis of conversation intelligence data, including call metrics, agent performance data, and customer satisfaction scores.
- [App Service](#) is a platform as a service (PaaS) offering for building and hosting web applications. In this architecture, App Service hosts both the orchestration APIs that coordinate data processing workflows and the interactive web front end that enables users to explore conversation insights through natural language interaction.
- [Azure OpenAI](#) is a cloud-based platform from Microsoft that provides access to advanced language models for natural language processing and generation. In this architecture, Azure OpenAI powers the conversational chat interface. This interface enables users to ask questions about their conversation data and receive contextual responses through the RAG pattern.
- [Semantic Kernel](#) is an open-source SDK that integrates large language models with conventional programming languages. In this architecture, Semantic Kernel orchestrates the interaction between Azure OpenAI and other Azure AI services. It also manages function calling and coordinates intelligent workflows.

- [Azure Cosmos DB](#) is a globally distributed, multiple-model database service that has guaranteed low latency. In this architecture, Azure Cosmos DB stores chat history and user session data. This storage enables fast access to conversation context for the interactive front-end experience.
- [Azure Container Registry](#) is a managed Docker registry service for storing and managing container images. In this architecture, Container Registry manages container images for the application components. This management ensures consistent deployment and version control across the solution.
- [Azure AI Foundry](#) is a unified PaaS offering from Microsoft that provides a comprehensive collection of AI capabilities through REST APIs and SDKs. In this architecture, Azure AI Foundry enhances conversation analysis via extra topic modeling capabilities. These capabilities complement core content understanding and search functionality.

Scenario details

This conversation knowledge mining solution addresses the challenge of extracting actionable insights from large volumes of unstructured conversational data that organizations accumulate from customer interactions, support calls, sales conversations, and internal meetings.

Traditional analysis methods struggle to process conversational data at scale and extract meaningful patterns. As a result, organizations face limitations in understanding customer sentiment, identifying operational problems, and uncovering opportunities for improvement.

The solution uses advanced AI capabilities to automatically process audio recordings and text transcripts, extract key entities and relationships, and create searchable knowledge bases that can be explored through natural language interaction. These functionalities enable analysts and business users to quickly identify trends, understand customer feedback patterns, and make data-driven decisions without requiring technical expertise in data analysis or query languages.

Potential use cases

Consider the following potential use cases.

Customer service optimization

- **Contact center quality improvement:** Analyze customer support calls to identify common problems, measure resolution effectiveness, and discover training opportunities for support agents across different product lines and service categories.

- **Agent performance evaluation:** Analyze conversation patterns to identify techniques that top-performing agents use, identify common resolution strategies, and highlight coaching opportunities. Track metrics such as first-call resolution rates and escalation patterns.
- **Customer sentiment analysis:** Extract emotional indicators and satisfaction trends from customer interactions to improve service delivery. Use these insights to identify at-risk accounts and refine strategies to enhance customer experience.
- **Support ticket correlation:** Connect conversation themes with support ticket outcomes to optimize routing and reduce resolution times. This approach helps proactively address recurring customer concerns.
- **Quality scoring automation:** Extract conversation elements that correlate with customer satisfaction scores. Use these insights to automate quality assurance processes and ensure consistent evaluation across all interactions.

Sales and marketing intelligence

- **Sales conversation analysis:** Extract insights from sales calls to identify successful conversation patterns, objection-handling techniques, and competitive intelligence. Use these insights to improve sales effectiveness and enhance training programs.

Alternatives

This architecture includes multiple components that you can substitute with other Azure services or approaches, depending on your workload's functional and nonfunctional requirements. Evaluate the following alternatives and trade-offs to align with your specific goals.

Content processing approach

- **Current approach:** This solution uses Content Understanding as the primary service for extracting entities, relationships, and insights from conversational data. It provides specialized conversation analysis capabilities with built-in understanding of dialogue patterns and conversational context.
- **Alternative approach:** Use Azure AI Document Intelligence combined with Azure OpenAI to process transcripts and extract structured information by using prompt engineering and few-shot learning (FSL) techniques.

Consider the alternative approach if your workload has the following characteristics:

- You need to process conversations that are already in well-structured document formats.
- You require custom entity extraction that goes beyond standard conversation analysis.
- You want more control over the extraction logic through custom prompts and examples.
- Your conversations include complex formatting or mixed content types that require document-specific processing.

Search and retrieval strategy

- **Current approach:** This solution uses AI Search with vector embeddings to enable semantic search across conversation transcripts. It supports natural language queries and RAG-based interactions.
- **Alternative approach:** Implement a pure vector database solution by using Azure DocumentDB with vector search capabilities. Alternatively, use Azure Database for PostgreSQL with the pgvector extension.

Consider the alternative approach if your workload has the following characteristics:

- You need extremely high-performance vector similarity search with minimal latency.
- Your solution requires tight integration with existing MongoDB or PostgreSQL ecosystems.
- You want to minimize the number of different services in your architecture.
- Your search requirements are primarily vector-based and don't need traditional full-text search capabilities.

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

For more information about the costs to run this scenario, see the preconfigured estimate in the [Azure pricing calculator](#).

Pricing varies by region and usage, so it's not possible to predict exact costs for your specific workload. Most Azure resources in this infrastructure follow usage-based pricing tiers. However, some services, such as Container Registry, incur fixed daily costs for each registry. Other services, like SQL Database and Azure Cosmos DB, might generate baseline charges as soon as they're provisioned, regardless of actual usage.

Deploy this scenario

To deploy an implementation of this architecture, follow the steps in the [GitHub repository](#).

The deployment includes automated infrastructure as code (IaC) templates, sample conversation data for testing, and setup documentation to help you get started.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal author:

- [Solomon Pickett](#) | Software Engineer II

Other contributor:

- [Malory Rose](#) | Senior Software Engineer

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next step

- [AI Search vector search capabilities](#)

Related resource

- [Design and develop a RAG solution](#)

Image classification on Azure

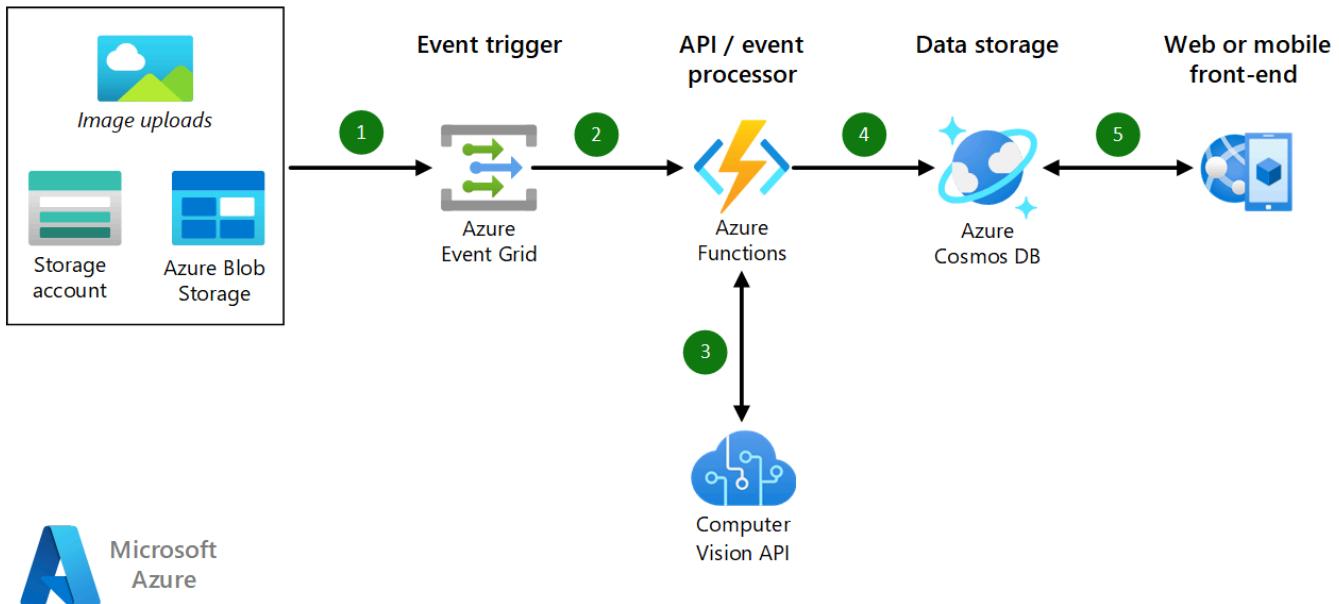
[Azure Blob Storage](#)[Azure Computer Vision](#)[Azure Cosmos DB](#)[Azure Event Grid](#)[Azure Functions](#)

Solution ideas

This article describes a solution idea. Your cloud architect can use this guidance to help visualize the major components for a typical implementation of this architecture. Use this article as a starting point to design a well-architected solution that aligns with your workload's specific requirements.

By using Azure services, such as the Computer Vision API and Azure Functions, companies can eliminate the need to manage individual servers, while reducing costs and utilizing the expertise that Microsoft has already developed with processing images with Azure AI services. This solution idea specifically addresses an image-processing use case. If you have different AI needs, consider the full suite of [Azure AI services](#).

Architecture



Download a [Visio file](#) of this solution idea.

Dataflow

This scenario covers the back-end components of a web or mobile application. Data flows through the scenario as follows:

1. Adding new files (image uploads) in Blob storage triggers an event in Azure Event Grid.
The uploading process can be orchestrated via the web or a mobile application.

- Alternatively, images can be uploaded separately to the Azure Blob storage.
2. Event Grid sends a notification that triggers the Azure functions.
 3. Azure Functions calls the Azure AI Vision API to analyze the newly uploaded image. Azure AI Vision accesses the image via the blob URL that's parsed by Azure Functions.
 4. Azure Functions persists the AI Vision API response in Azure Cosmos DB. This response includes the results of the analysis, along with the image metadata.
 5. The results can be consumed and reflected on the web or mobile front end. This approach retrieves the results of the classification but not the uploaded image.

Components

- [Azure AI Vision](#) is part of the Azure AI services suite. In this architecture, it retrieves information about each image. It analyzes newly uploaded images and provides metadata and classification results. These results enable automated image understanding.
- [Azure Functions](#) is a serverless solution that you can use to build robust apps with less code and less infrastructure. In this architecture, Azure Functions provides the back-end API for the web application. This platform also provides event processing for uploaded images. Azure Functions orchestrates workflow steps, including calling the AI Vision API, processing analysis results, and persisting metadata in the database.
- [Azure Event Grid](#) is a managed event-routing service that enables uniform event consumption by using a publish-subscribe model. In this architecture, Azure Event Grid triggers an event when a new image is uploaded to blob storage and initiates automated processing workflows by alerting Azure Functions of new uploads.
- [Azure Blob Storage](#) is an object storage solution for storing unstructured data in the cloud. In this architecture, it stores all of the image files that are uploaded into the web application, as well any static files that the web application consumes. Blob Storage is the primary repository for incoming image data, serving as both the source for processing and a reference for image access.
- [Azure Cosmos DB](#) is a NoSQL database. In this architecture, Azure Cosmos DB stores metadata about each image that is uploaded, including the results of the processing from Computer Vision API.

Alternatives

- [Azure OpenAI GPT-4o and GPT-4o-mini](#). GPT-4o and GPT-4o-mini are multimodal chat models from OpenAI that can answer general questions about what's present in the images you provide.

- [Custom Vision Service](#). The Computer Vision API returns a set of [taxonomy-based categories](#). If you need to process information that isn't returned by the Computer Vision API, consider the Custom Vision Service, which lets you build custom image classifiers. To learn about this service, follow the quick start [Build an image classification model with the Custom Vision](#).
- [Azure AI Search](#). If your use case involves querying the metadata to find images that meet specific criteria, consider using Azure AI Search.
- [Logic Apps](#). If you don't need to react in real-time on added files to a blob, you might consider using Logic Apps. A logic app which can check if a file was added might be started by the [recurrence trigger or sliding windows trigger](#).
- If you have images embedded in documents, use [Azure AI Document Intelligence](#) to locate those images. With that information, you can extract and perform further computer vision tasks on the embedded images. Use Document Intelligence to gather data about those embedded images, such as page number or caption text which can be stored along with the images' other metadata received through the Computer Vision API. If your images are mainly photos or scans of documents, use the [Document Intelligence custom classification models](#) to perform classification of an input file one page at a time to identify the documents within. This approach can also identify multiple documents or multiple instances of a single document within an input file.

Scenario details

This scenario is relevant for businesses that need to process images.

Potential applications include classifying images for a fashion website, analyzing text and images for insurance claims, or understanding telemetry data from game screenshots. Traditionally, companies would need to develop expertise in machine learning models, train the models, and finally run the images through their custom process to get the data out of the images.

Potential use cases

This solution is ideal for the retail, game, finance, and insurance industries. Other relevant use cases include:

- **Classifying images on a fashion website.** Image classification can be used by sellers while uploading pictures of products on the platform for sale. They can then automate the consequent manual tagging involved. The customers can also search through the visual impression of the products.

- **Classifying telemetry data from screenshots of games.** The classification of video games from screenshots is evolving into a relevant problem in social media, coupled with computer vision. For example, when Twitch streamers play different games in succession, they might skip manually updating their stream information. Failure to update stream information could result in the misclassification of streams in user searches and might lead to the loss of potential viewership for both the content creators and the streaming platforms. While introducing novel games, a custom model route could be helpful to introduce the capability to detect novel images from those games.
- **Classifying images for insurance claims.** Image classification can help reduce the time and cost of claims processing and underwriting. It could help analyze natural-disaster damage, vehicle-damage, and identify residential and commercial properties.

Next steps

Product documentation

- [What is Azure AI Vision?](#)
- [AI enrichment in Azure AI Search](#)
- [Introduction to Azure Functions](#)
- [What is Azure Event Grid?](#)
- [Introduction to Azure Blob storage](#)
- [Welcome to Azure Cosmos DB](#)

For a guided learning path, see:

- [Build a serverless web app in Azure](#)
- [Classify images with Azure AI Custom Vision](#)
- [Use AI to recognize objects in images by using the Custom Vision service](#)
- [Classify endangered bird species with Custom Vision](#)
- [Classify images with Azure AI Custom Vision services](#)
- [Detect objects in images with Azure AI Custom Vision](#)

Related resources

- [Use AI enrichment with image and text processing](#)
- [Get started with multimodal vision chat apps using Azure OpenAI](#)

Use AI to forecast customer orders

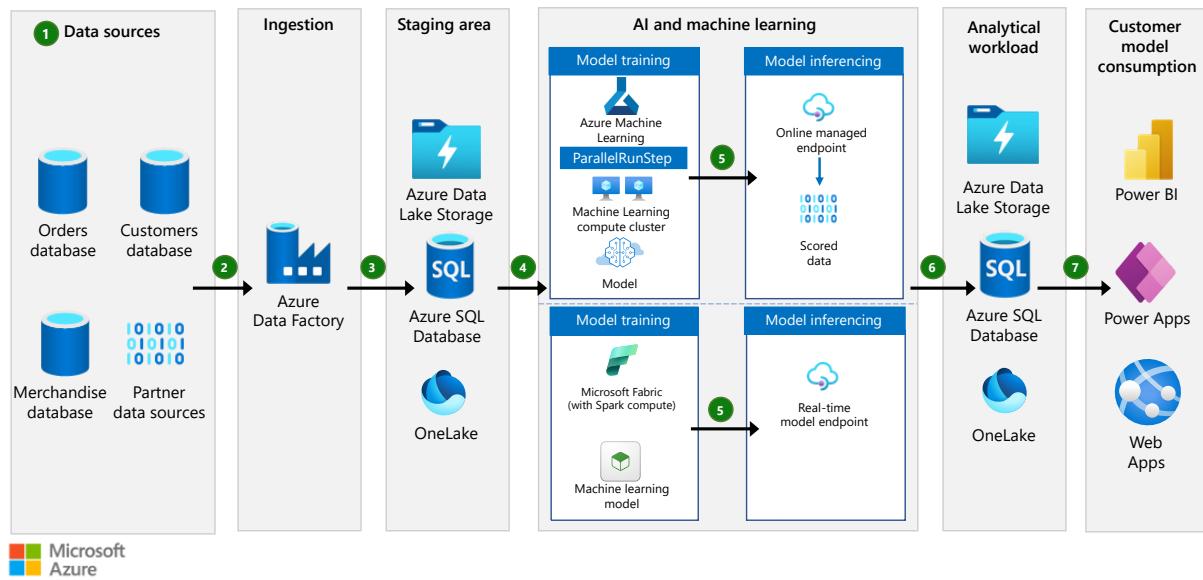
[Azure Machine Learning](#)[Microsoft Fabric](#)[Azure Data Lake](#)[Azure SQL Database](#)[Power Apps](#)

Solution ideas

This article describes a solution idea. Your cloud architect can use this guidance to help visualize the major components for a typical implementation of this architecture. Use this article as a starting point to design a well-architected solution that aligns with your workload's specific requirements.

This article describes how merchandise distributors can use AI and machine learning to predict a customer's future order quantity for a specific stock-keeping unit (SKU). Distributors use next-order forecasting to provide product recommendations and suggest optimal order quantities to customers. This article builds on the concepts described in the [many-models machine learning architecture](#).

Architecture



Download a [PowerPoint file](#) of this architecture.

Dataflow

1. Data sources

To forecast future orders, you need comprehensive data about your customers' buying history for various SKUs at specific stores, including information about preferences and purchasing behavior. This kind of information is typically obtained from orders, merchandise, and customer databases. You also need to consider external factors like weather, holidays, and events. This data is usually obtained from partner sources.

To create order forecasting models, you use data in a schema that includes several key variables:

- Date and time
- Customer store location
- Merchandise SKU
- Quantity ordered
- Price per unit
- Weather-related features, holidays, events, and other external factors

By analyzing this data, you can gain insights into customer behavior and make informed SKU and quantity recommendations for the customer's next order.

2. Ingestion

Data ingestion is the process of transferring data from various sources to a designated destination. This process uses specific connectors for each data source and target destination.

Azure Data Factory provides connectors that you can use to extract data from various sources, including databases, file systems, and cloud services. These connectors are created by Microsoft or by partner vendors and are designed to function effectively with multiple data sources. For example, you can use [SAP connectors](#) for various SAP data ingestion scenarios. You can use the [Snowflake connector](#) to copy data from Snowflake.

3. Staging area

The staging area serves as a temporary storage location between the source and the destination. The main purpose of this staging area is to maintain data in a uniform and structured format while it undergoes transformations or quality checks before it's loaded into its destination.

A consistent data format is critical for accurate analysis and modeling. If you consolidate and prepare the data in a staging area, Azure Machine Learning can process it more efficiently.

4. Machine learning model training

Model training is a machine learning process that uses an algorithm to learn patterns from data and, in this solution, select a model that can accurately predict a customer's next order.

In this solution, you can use Machine Learning or Fabric Data Science to train models.

- [Machine Learning](#) is used to manage the entire machine learning project life cycle, including training models, deploying models, and managing machine learning operations.
 - [ParallelRunStep](#) is used to process large amounts of data in parallel and create models that can forecast the next order for every customer store and merchandise SKU combination. You can reduce processing time by dividing the dataset into smaller parts and processing them simultaneously on multiple virtual machines. You can use Machine Learning compute clusters to distribute workloads across multiple nodes.
 - After the data is prepared, Machine Learning can start the parallel model training process by using ParallelRunStep with a range of forecasting models, including exponential smoothing, elastic net, and Prophet. Each node or compute instance starts building the model, so the process is more efficient and faster.
- Apache Spark, a part of Microsoft Fabric, enables machine learning with big data. By using Apache Spark, you can build valuable insights into large masses of structured, unstructured, and fast-moving data. You have several available open-source library options when you [train machine learning models with Apache Spark in Fabric](#), including Apache Spark MLlib and [SynapseML](#).

5. Machine learning model inferencing

Model inferencing is a process that uses a trained machine learning model to generate predictions for previously unseen data points. In this solution, it forecasts the quantity of the merchandise SKU that a customer is likely to purchase.

- Machine Learning provides model registries for storing and versioning trained models. Model registries can help you organize and track trained models and ensure that they're readily available for deployment.
 - Deploying a trained machine learning model enables the model to process new data for inferencing. We recommend that you use [Azure managed endpoints](#) for the deployment target. Endpoints enable easy scalability, performance tuning, and high availability.

In this use case, there are two ways to deploy models on the [managed endpoints](#). The first option is to deploy each model on its own managed endpoint, as shown in the diagram. The second option is to bundle multiple models into a single model and deploy it on a single managed endpoint. The latter approach is more efficient and provides an easier way to deploy and manage multiple models simultaneously.

- Fabric provides real-time predictions from machine learning models by using secure, scalable, and easy-to-use online endpoints. These endpoints are available as built-in properties of most Fabric models.
 - You can activate, configure, and query model endpoints by using a [public-facing REST API](#). You can get started from the Fabric interface by using a low-code experience to activate model endpoints and preview predictions.

6. Analytical workload

The output of the model is stored in analytics systems like OneLake, Azure Data Lake Storage, or Azure SQL Database, where the input data is also collected and stored. This stage facilitates the availability of the prediction results for customer consumption, model monitoring, and new data for retraining models to improve their accuracy.

7. Customer consumption

To visually present the scored model to customers, you can use the Web Apps feature of Azure App Service, a Power BI dashboard, or Power Apps. These platforms allow you to display SKU recommendations and predicted quantities in a clear, graphical format.

Customers are notified of recommended SKUs and predicted quantities, so they can place orders proactively. The recommendations can help streamline the ordering process, reduce the likelihood of stockouts, and enhance customer satisfaction. If you use a Power BI dashboard or Power Apps, you can provide an informative ordering experience to your customers.

Components

- [Fabric](#) is an enterprise-ready, end-to-end analytics platform. It unifies data movement, data processing, ingestion, transformation, real-time event routing, and report building. It supports these capabilities with integrated services like Fabric Data Engineer, Azure Data Factory, Data Science, Fabric Real-Time Intelligence, Fabric Data Warehouse, and Fabric Databases. In this architecture, Fabric workloads are used to process and transform data to create end-to-end data science workflows.

- **OneLake** is a single, unified, logical data lake for the whole organization. Every Fabric tenant automatically includes OneLake with no infrastructure to manage. In this architecture, OneLake is used to store data that's ingested from various sources. It also serves as a repository for data that trains machine learning models. This centralized storage system ensures efficient data management and access for analytical processes.
 - A **Fabric lakehouse** is a data architecture platform for storing, managing, and analyzing structured and unstructured data in a single location. It's a flexible and scalable solution that allows organizations to handle large volumes of data by using various tools and frameworks to process and analyze that data. It integrates with other data management and analytics tools to provide a solution for data engineering and analytics. A lakehouse combines the scalability of a data lake with the performance and structure of a data warehouse to provide a unified platform for data storage, management, and analytics. In this architecture, the Fabric lakehouse is a central hub for both structured and unstructured data from diverse sources. It helps you ensure that you have a thorough dataset. It also stores model outputs so that applications can easily access and consume prediction results.
 - **Data Warehouse** is the data warehousing solution within Fabric. The lake-centric warehouse is built on a distributed processing engine that enables industry-leading performance at scale while minimizing the need for configuration and management. In this architecture, you use Data Warehouse to store transformed and unified data during the analytical workload phase instead of relying on a lakehouse. Use this [decision guide](#) to help you choose a data store for your Fabric workloads.
- **Azure Data Factory** is a cloud-based data integration service that automates data movement and transformation. In this architecture, Azure Data Factory ingests data from diverse sources and moves it through the pipeline for processing and analysis.
 - **Data Lake Storage** is a limitless data storage service for housing data in various shapes and formats. It provides easy integration with the analytics tools in Azure. This solution uses a local data store for machine learning data and a premium data cache for training the machine learning model.
 - **Machine Learning** is an enterprise-grade machine learning service that facilitates model development and deployment to a wide range of machine learning compute targets. In this architecture, it provides users at all skill levels with a low-code designer, automated machine learning, and a hosted Jupyter Notebook environment that supports various integrated development environments.
 - **Machine Learning compute clusters** are managed compute structures that you can use to easily create single-node or multiple-node compute resources. In this architecture,

compute clusters enable parallel processing to train multiple forecasting models simultaneously for every customer store and merchandise SKU combination.

- [Machine Learning endpoints](#) are HTTPS endpoints that clients can call to receive the inferencing (scoring) output of a trained model. An endpoint provides a stable scoring URI that's authenticated via key-and-token authentication. In this architecture, endpoints enable applications to get real-time predictions for customer order quantities.
- [Machine Learning pipelines](#) are independently executable workflows of complete machine learning tasks. In this architecture, pipelines can help you standardize the best practices of producing a machine learning model and improve model-building efficiency.
- [SQL Database](#) is an always-up-to-date, fully managed relational database service that's built for the cloud. In this architecture, it stores structured data, such as order history and model outputs, to support analytical workloads and reporting.
- [Power BI](#) provides business analytics and visually immersive and interactive insights. In this architecture, Power BI provides dashboards and reports that present insights and recommendations that the machine learning models generate.
- [Power Apps](#) is a platform for building custom business applications quickly. In this architecture, use Power Apps to create user-facing applications that display personalized order recommendations. Data can be stored in the underlying data platform ([Microsoft Dataverse](#)) or in various online and on-premises data sources, like SharePoint, Microsoft 365, Dynamics 365, and SQL Server.
- [Web applications](#) that are built with ASP.NET Core and hosted in Azure provide competitive advantages over traditional alternatives. ASP.NET Core is optimized for modern web application development practices and cloud hosting scenarios. In this architecture, web applications can serve as portals for users to access forecasted recommendations and interact with the ordering system.

Alternatives

- Machine Learning provides data modeling and deployment in this solution. Alternatively, you can use [Data Science](#) experiences that empower users to build end-to-end data science workflows. You can complete a wide range of activities across the entire data science process. Your choice between Machine Learning and Fabric depends on factors such as the scale of your data science operations, the complexity of your machine learning tasks, and integration with other tools and services that you already use. Both

platforms provide coverage across a range of requirements and features, which makes them suitable for a wide range of scenarios.

You can also use Azure Databricks to explore and manipulate data in this solution. Azure Databricks uses a notebook-based interface that supports the use of Python, R, Scala, and SQL. Azure Databricks mainly provides data processing and analysis capabilities.

Use [OneLake](#) to mount your existing platform as a service (PaaS) storage accounts by using the [shortcut](#) feature. You don't migrate or copy your existing data. Shortcuts provide direct access to data in Data Lake Storage. They also enable data sharing between users and applications without duplicating files. You can also create shortcuts to other storage systems so that you can analyze cross-cloud data with intelligent caching that reduces egress costs and brings data closer to compute.

[Fabric Data Factory](#) provides a modern data integration experience to ingest, prepare, and transform data from a rich set of data sources. It incorporates the simplicity of Power Query, and you can use more than 200 native connectors to connect to data sources on-premises and in the cloud. You can use Fabric data pipelines instead of Azure Data Factory pipelines for data integration, depending on several factors. For more information, see [Compare Azure Data Factory and Fabric Data Factory](#).

Databases in Fabric are developer-friendly, transactional databases like SQL Database that help you create your operational database in Fabric. Use the mirroring capability to bring data from various systems together into OneLake. You can continuously replicate your existing data estate directly into OneLake, including data from SQL Database, Azure Cosmos DB, Azure Databricks, Snowflake, and Fabric SQL databases. For more information, see [SQL database in Fabric](#) and [Mirroring in Fabric](#).

- Power BI is a popular tool for visualization. Grafana is another option. The main difference is that Grafana is open source, while Power BI is a software as a service (SaaS) offering from Microsoft. If you prioritize customization and the use of open-source tools, Grafana is a better choice. If you prioritize integration with other Microsoft products and product support, Power BI is a better choice.
- Instead of using an endpoint for each model, you can bundle multiple models into a single model for deployment to a single managed endpoint. Bundling models for deployment is known as *model orchestration*. Potential drawbacks of using this approach include increased complexity, potential conflicts between models, and increased risk of downtime if the single endpoint fails.

Scenario details

The merchandise distribution industry historically struggles to gain insights into customer behavior and purchasing patterns, which makes it difficult to provide personalized product recommendations, improve customer satisfaction, and drive sales. By using AI and machine learning, merchandise distributors can transform the industry.

By adopting next-order forecasting, organizations can recommend products and quantities based on customer purchasing patterns. This methodology benefits customers by consolidating orders and reducing transportation and logistics costs. It also allows distributors to establish smart contracts with regular customers. These contracts enable distributors to proactively recommend products and quantities at a regular cadence, manage inventory, influence manufacturing efficiencies, save money, and promote sustainability. For example, by implementing accurate forecasting, distributors of perishable items can manage optimum levels of inventory and avoid dumping excess stock into landfills.

Next-order forecasting uses AI and machine learning algorithms to analyze customer orders and make recommendations for future orders. The architecture described in this article takes next-order forecasting to another level by enabling forecasting at the individual SKU and store level by using parallel processing. This combination enables businesses to forecast demand for specific products at specific stores. By using this methodology, you can provide your customers with personalized recommendations that meet their needs and exceed their expectations.

Potential use cases

Organizations can use next-order forecasting to predict customer demand and optimize inventory management. The following examples are some specific use cases:

- **E-commerce:** Online retailers can forecast customer demand and recommend products based on customer purchase history, browsing behavior, and preferences. These predictions can improve the customer experience, increase sales, and reduce the cost of logistics and warehousing.
- **Hospitality:** Hotels and restaurants can predict customer demand for menu items, beverages, and other products. These predictions can help them optimize inventory, reduce food waste, and improve profitability.
- **Healthcare:** Hospitals and clinics can forecast patient demand for medical supplies, equipment, and medications. These forecasts can help them reduce inventory stockouts, avoid overstocking, and optimize procurement processes.
- **Manufacturing:** Manufacturers can forecast demand for products and raw materials, optimize inventory levels, and improve supply chain resilience.

- **Energy:** Energy companies can predict demand and optimize energy generation, transmission, and distribution. Next-order forecasting can help them reduce their carbon footprint and improve sustainability.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Well-Architected Framework](#).

The technologies in this solution were chosen for scalability, availability, and cost optimization.

Security

Security provides assurances against deliberate attacks and the misuse of your valuable data and systems. For more information, see [Design review checklist for Security](#).

Improved security is built in to the components of this scenario. You can use Microsoft Entra authentication or role-based access control to manage permissions. Consider implementing [Machine Learning best practices for enterprise security](#) to establish appropriate security levels.

Fabric provides a complete [security](#) package for the entire platform. Fabric minimizes the cost and responsibility of maintaining your security solution and transfers it to the cloud. By using Fabric, you can access the expertise and resources of Microsoft to keep your data more secure, patch vulnerabilities, monitor threats, and comply with regulations. You can also use Fabric to manage, control, and audit your security settings according to your requirements.

Data Lake provides improved data protection, data masking, and improved threat protection. For more information, see [Data Lake security](#).

For more information about security for this architecture, see the following resources:

- [Integrate Azure services with virtual networks for network isolation](#)
- [Enterprise security and governance for Machine Learning](#)

Operational Excellence

Operational Excellence covers the operations processes that deploy an application and keep it running in production. For more information, see [Design review checklist for Operational Excellence](#). Observability, monitoring, and diagnostic settings are important considerations to highlight under this pillar.

Observability refers to the ability to understand how the data flow of a system functions.

Monitoring is the ongoing process of tracking the performance of a system over time. You can monitor metrics like CPU usage, network traffic, and response times. *Diagnostic settings* are configuration options that you can use to capture diagnostic information.

For more information, see [Overview of the Operational Excellence pillar](#).

Follow [machine learning operations guidelines](#) to manage an end-to-end machine learning life cycle that's scalable across multiple workspaces. Before you deploy your solution to the production environment, make sure that it supports ongoing inference with retraining cycles and automated redeployment of models.

For more information, see the following resources:

- [Machine learning operations v2](#)
- [Azure machine learning operations v2 GitHub repository](#) ↗

Performance Efficiency

Performance Efficiency refers to your workload's ability to scale to meet user demands efficiently. For more information, see [Design review checklist for Performance Efficiency](#).

Most components in this architecture can be scaled up and down based on the analysis activity levels. The [Fabric Capacity Metrics app](#) is designed to provide monitoring capabilities for Fabric capacities. Use the app to monitor your capacity consumption and make informed decisions about how to use your capacity resources. For example, the app can help identify when to scale up your capacity or when to turn on autoscale.

You can scale [Machine Learning](#) based on the amount of data and the compute resources that you need for model training. You can scale the deployment and compute resources based on the expected load and scoring service.

Load testing is an important step for ensuring the performance efficiency of the machine learning model. This testing involves the simulation of a high volume of requests to the model to measure metrics like throughput, response time, and resource utilization. Load testing can help you identify bottlenecks and problems that can affect the model's performance in a production environment.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal author:

- [Manasa Ramalinga](#) | Principal Cloud Solution Architect – US Customer Success

Other contributors:

- [Oscar Shimabukuro Kiyano](#) | Senior Cloud Solution Architect – US Customer Success
- [Veera Vemula](#) | Senior Cloud Solution Architect – US Customer Success

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- [What is Machine Learning?](#)
- [Track experiments and models by using MLflow](#)
- [Azure Data Factory documentation](#)
- [What is Fabric?](#)
- [What is SQL Database?](#)
- [What is Power BI?](#)
- [Overview of Azure Stream Analytics](#)

Related resources

- [Many-models machine learning at scale with Machine Learning](#)
- [Machine learning operations](#)

Use Azure Databricks to orchestrate MLOps

Azure Databricks

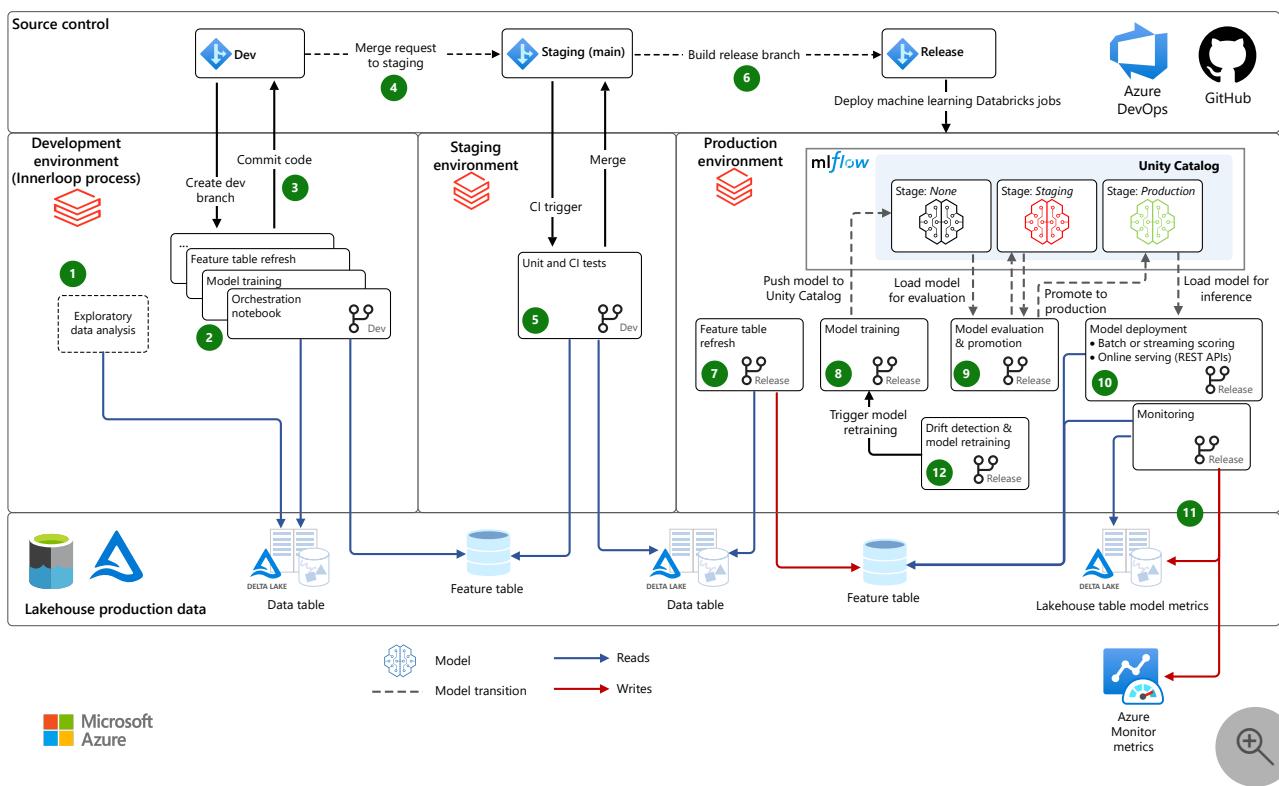
Solution ideas

This article describes a solution idea. Your cloud architect can use this guidance to help visualize the major components for a typical implementation of this architecture. Use this article as a starting point to design a well-architected solution that aligns with your workload's specific requirements.

This article provides a [machine learning operations \(MLOps\)](#) architecture and process that uses Azure Databricks. Data scientists and engineers can use this standardized process to move machine learning models and pipelines from development to production.

This solution can take advantage of full automation, continuous monitoring, and robust collaboration. As a result, it targets level 4 of MLOps maturity. This architecture uses the *promote code that generates the model* approach rather than the *promote models* approach. The *promote code that generates the model* approach focuses on writing and managing the code that generates machine learning models. The recommendations in this article include options for automated or manual processes.

Architecture



[Download a Visio file](#) of this architecture.

Workflow

The following workflow corresponds to the previous diagram. Use source control and storage components to manage and organize code and data.

Source control: This project's code repository organizes the notebooks, modules, and pipelines. You can create development branches to test updates and new models. Develop code in Git-supported notebooks or integrated development environments (IDEs) that integrate with [Git folders](#) so that you can sync with your Azure Databricks workspaces. Source control promotes machine learning pipelines from the development environment, to testing in the staging environment, and to deployment in the production environment.

Lakehouse production data: As a data scientist, you have read-only access to production data in the development environment. The development environment can have mirrored data and redacted confidential data. You also have read and write access in a dev storage environment for development and experimentation. We recommend that you use a [lakehouse](#) architecture for data in which you store [Delta Lake](#)-format data in [Azure Data Lake Storage](#). A lakehouse provides a robust, scalable, and flexible solution for data management. To define access controls, use [table access controls](#).

The main workflow consists of the following environments.

Development

In the development environment, you develop machine learning pipelines.

1. **Do exploratory data analysis (EDA).** Explore data in an interactive, iterative process. You might not deploy this work to staging or production. Use tools like [Databricks SQL](#), the `dbutils.data.summarize` command, and [Databricks AutoML](#).
2. **Develop model training and other machine learning pipelines.** Develop machine learning pipelines modular code, and orchestrate code via Databricks notebooks or an MLflow project. In this architecture, the model training pipeline reads data from the feature store and other lakehouse tables. The pipeline trains and tunes log model parameters and metrics to the [MLflow tracking server](#). The [feature store API](#) logs the final model. These logs include the model, its inputs, and the training code.
3. **Commit code.** To promote the machine learning workflow toward production, commit the code for featurization, training, and other pipelines to source control. In the code base, place machine learning code and operational code in different folders so that team members can develop code at the same time. Machine learning code is code that's

related to the model and data. Operational code is code that's related to Databricks jobs and infrastructure.

This core cycle of activities that you do when you write and test code are referred to as the *innerloop process*. To carry out the innerloop process for the development phase, use Visual Studio Code (VS Code) in combination with the dev container command-line interface (CLI) and the Databricks CLI. You can write the code and do unit testing locally. You should also submit, monitor, and analyze the model pipelines from the local development environment.

Staging

In the staging environment, continuous integration (CI) infrastructure tests changes to machine learning pipelines in an environment that mimics production.

4. **Merge a request.** When you submit a merge request or pull request against the staging (or *main*) branch of the project in source control, a continuous integration and continuous delivery (CI/CD) tool like [Azure DevOps](#) runs tests.
5. **Run unit tests and CI tests.** Unit tests run in CI infrastructure, and integration tests run in end-to-end [workflows](#) on Azure Databricks. If tests pass, the code changes merge.
6. **Build a release branch.** When you want to deploy the updated machine learning pipelines to production, you can build a new release. A deployment pipeline in the CI/CD tool redeploys the updated pipelines as new [workflows](#).

Production

Machine learning engineers manage the production environment, where machine learning pipelines directly serve end applications. The key pipelines in production refresh feature tables, train and deploy new models, run inference or serving, and monitor model performance.

7. **Feature table refresh:** This pipeline reads data, computes features, and writes to [feature store](#) tables. You can set up this pipeline to either run continuously in streaming mode, run on a schedule, or run on a trigger.
8. **Model training:** In production, you can set up the model training or retraining pipeline to either run on a trigger or a schedule to train a fresh model on the latest production data. Models automatically register to [Unity Catalog](#).
9. **Model evaluation and promotion:** When a new model version is registered, the CD pipeline starts, which runs tests to ensure that the model performs well in production. When the model passes tests, Unity Catalog tracks its progress via model stage transitions. Tests include compliance checks, A/B tests to compare the new model with

the current production model, and infrastructure tests. Lakehouse tables record test results and metrics. You can optionally require manual sign-offs before models transition to production.

10. Model deployment: When a model enters production, it's deployed for scoring or serving. The most common deployment modes include the following options:

- *Batch or streaming scoring:* For latencies of minutes or longer, [batch and streaming](#) are the most cost-effective options. The scoring pipeline reads the latest data from the feature store, loads the latest production model version from Unity Catalog, and performs inference in a Databricks job. It can publish predictions to lakehouse tables, a Java Database Connectivity (JDBC) connection, flat files, message queues, or other downstream systems.
- *Online serving (REST APIs):* For low-latency use cases, you generally need online serving. MLflow can deploy models to [Mosaic AI Model Serving](#), cloud provider serving systems, and other systems. In all cases, the serving system initializes with the latest production model from Unity Catalog. For each request, it fetches features from an online feature store and makes predictions.

11. Monitoring: Continuous or periodic [workflows](#) monitor input data and model predictions for drift, performance, and other metrics. You can use the [Lakeflow Declarative Pipelines](#) framework to automate monitoring for pipelines and store the metrics in lakehouse tables. [Databricks SQL](#), [Power BI](#), and other tools can read from those tables to create dashboards and alerts. To monitor application metrics, logs, and infrastructure, you can also integrate Azure Monitor with Azure Databricks.

12. Drift detection and model retraining: This architecture supports both manual and automatic retraining. Schedule retraining jobs to keep models fresh. After a detected drift crosses a preconfigured threshold that you set in the monitoring step, the retraining pipelines analyze the drift and trigger retraining. You can set up pipelines to start automatically, or you can receive a notification and then run the pipelines manually.

Components

- A [data lakehouse](#) architecture unifies the elements of data lakes and data warehouses. This architecture uses a lakehouse to get data management and performance capabilities that you typically find in data warehouses but with the low-cost, flexible object stores that data lakes provide.

We recommend [Delta Lake](#) as the open-source data format for a lakehouse. In this architecture, Delta Lake stores all machine learning data in Data Lake Storage and provides a high-performance query engine.

- [MLflow](#) is an open-source project for managing the end-to-end machine learning life cycle. In this architecture, MLflow tracks experiments, manages model versions, and facilitates model deployment to various inference platforms. MLflow has the following components:
 - The [tracking feature](#) in MLflow is a system for logging and managing machine learning experiments. In this architecture, it records and organizes parameters, metrics, and model artifacts for each experiment run. This capability enables you to compare results, reproduce experiments, and audit model development.
 - [Databricks autologging](#) is an automation feature that extends [MLflow automatic logging](#) to track machine learning experiments by capturing model parameters, metrics, files, and lineage information. In this architecture, Databricks autologging ensures consistent experiment tracking and reproducibility by automatically recording these details.
 - An [MLflow model](#) is a standardized packaging format. In this architecture, MLflow models support model storage and deployment across different serving and inference platforms.
 - [Unity Catalog](#) is a data governance solution that provides centralized access control, auditing, lineage, and data-discovery capabilities across Azure Databricks workspaces. In this architecture, it governs access, maintains lineage, and structures models and data across workspaces.
 - [Mosaic AI Model Serving](#) is a service that hosts MLflow models as REST endpoints. In this architecture, it enables deployed machine learning models to serve predictions through APIs.
- [Azure Databricks](#) is a managed platform for analytics and machine learning. In this architecture, Azure Databricks integrates with enterprise security, provides high availability, and connects MLflow and other machine learning components for end-to-end MLOps.
 - [Databricks Runtime for Machine Learning](#) is a preconfigured environment that automates the creation of a cluster that's optimized for machine learning and preinstalls popular machine learning libraries like TensorFlow, PyTorch, and XGBoost. It also preinstalls Azure Databricks for Machine Learning tools, like AutoML and feature store clients. In this architecture, it provides ready-to-use clusters with popular machine learning libraries and tools.
 - A [feature store](#) is a centralized repository of features. In this architecture, the feature store supports feature discovery and sharing, and helps prevent data skew between model training and inference.

- [Databricks SQL](#) is a serverless data warehouse that integrates with different tools so that you can author queries and dashboards in your preferred environments without adjusting to a new platform. In this architecture, Databricks SQL lets you query data and create dashboards for analysis and reporting.
- [Git folders](#) are integrated workspace directories. In this architecture, Git folders connect Azure Databricks workspaces to your Git provider. This integration improves notebook or code collaboration and IDE integration.
- [Workflows](#) and [jobs](#) provide a way to run non-interactive code in an Azure Databricks cluster. In this architecture, workflows and jobs provide automation for data preparation, featurization, training, inference, and monitoring.

Alternatives

You can tailor this solution to your Azure infrastructure. Consider the following customizations:

- Use multiple development workspaces that share a common production workspace.
- Exchange one or more architecture components for your existing infrastructure. For example, you can use [Azure Data Factory](#) to orchestrate Databricks jobs.
- Integrate with your existing CI/CD tooling via Git and Azure Databricks REST APIs.
- Use [Microsoft Fabric](#) as an alternative service for machine learning capabilities. Fabric provides integrated workloads for data engineering (lakehouses with Apache Spark), data warehousing, and OneLake for unified storage.

Scenario details

This solution provides a robust MLOps process that uses Azure Databricks. You can replace all elements in the architecture, so you can integrate other Azure services and partner services as needed. This architecture and description are adapted from the e-book [The Big Book of MLOps: Second Edition](#). The e-book explores this architecture in more detail.

MLOps helps reduce the risk of failures in machine learning and AI systems and improve the efficiency of collaboration and tooling. For an introduction to MLOps and an overview of this architecture, see [Architect MLOps on the lakehouse](#).

Use this architecture to take the following actions:

- **Connect your business stakeholders with machine learning and data science teams.** Use this architecture to incorporate notebooks and IDEs for development. Business

stakeholders can view metrics and dashboards in Databricks SQL, all within the same lakehouse architecture.

- **Focus your machine learning infrastructure around data.** This architecture treats machine learning data just like other data. Machine learning data includes data from feature engineering, training, inference, and monitoring. This architecture reuses tooling for production pipelines, dashboarding, and other general data processing for machine learning data processing.
- **Implement MLOps in modules and pipelines.** As with any software application, use the modularized pipelines and code in this architecture to test individual components and decrease the cost of future refactoring.
- **Automate your MLOps processes as needed.** In this architecture, you can automate steps to improve productivity and reduce the risk of human error, but you don't need to automate each step. Azure Databricks permits user interface (UI) and manual processes, as well as APIs for automation.

Potential use cases

This architecture applies to all types of machine learning, deep learning, and advanced analytics. This architecture uses the following common machine learning and AI techniques:

- Classical machine learning, like linear models, tree-based models, and boosting
- Modern deep learning, like TensorFlow and PyTorch
- Custom analytics, like statistics, Bayesian methods, and graph analytics

The architecture supports both small data on a single machine and large data by using distributed computing and graphics processing unit (GPU)-accelerated resources. At each stage of the architecture, you can choose compute resources and libraries to adapt to your scenario's data size and problem dimensions.

The architecture applies to all types of industries and business use cases. Azure Databricks customers that use this architecture include small and large organizations in the following industries:

- Consumer goods and retail services
- Financial services
- Healthcare and life sciences
- Information technology

For more information, see [Databricks customers](#).

Foundation model fine-tuning in MLOps workflows

As more organizations use large language models for specialized tasks, they must add foundation model fine-tuning to the MLOps process. You can use Azure Databricks to fine-tune foundation models with your data. This capability supports custom applications and a mature MLOps process. In the context of the MLOps architecture in this article, fine-tuning aligns with several best practices:

- **Modularized pipelines and codes:** Fine-tuning tasks can be encapsulated as modular components within the training pipeline. This structure enables isolated evaluation and simplifies refactoring.
- **Experiment (fine-tuning run) tracking:** MLflow integration logs each fine-tuning run with specific parameters like the number of epochs and learning rate, and with metrics like loss and cross-entropy. This process improves reproducibility, auditability, and the ability to measure improvements.
- **Model registry and deployment:** Fine-tuned models are automatically registered in Unity Catalog. This automation supports deployment and governance.
- **Automation and CI/CD:** Fine-tuning jobs can be initiated via Databricks workflows or CI/CD pipelines. This process supports continuous learning and model refresh cycles.

This approach lets teams maintain high MLOps maturity while using the flexibility and power of foundation models. For more information, see [Foundation model fine-tuning](#).

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal authors:

- [Brandon Cowen](#) | Senior Cloud Solution Architect
- [Prabal Deb](#) | Principal Software Engineer

Other contributors:

- [Rodrigo Rodríguez](#) | Senior Cloud Solution Architect, AI & Quantum

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- [AI and machine learning on Databricks](#)

- [Databricks machine learning product page and resources](#)
- [Train AI and machine learning models on Azure Databricks](#)

Related resources

- [MLOps maturity model](#)
- [MLOps v2](#)

Use the many-models architecture approach to scale machine learning models

Azure Data Factory

Azure Data Lake

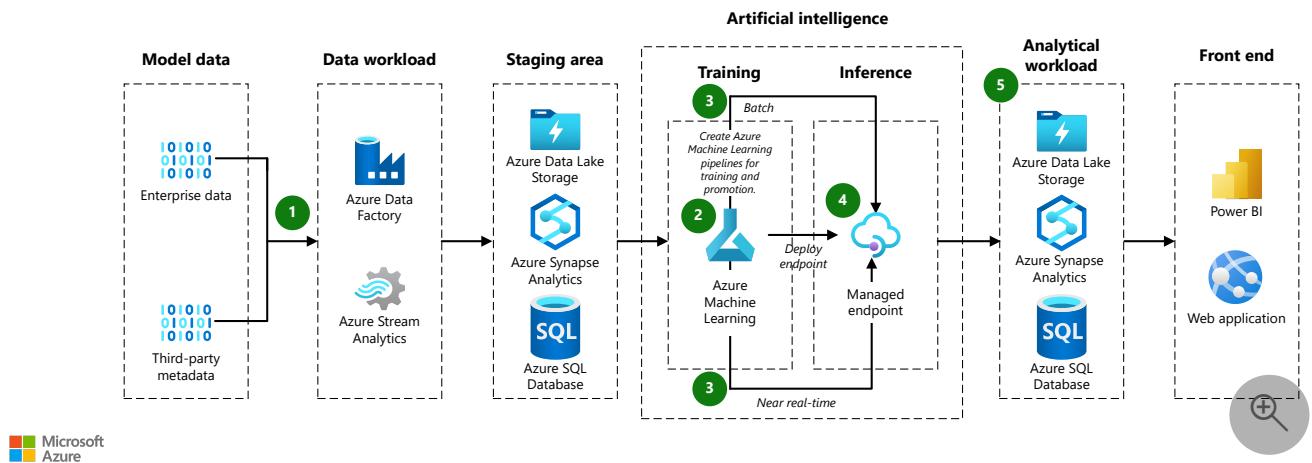
Azure Databricks

Azure Machine Learning

Azure Synapse Analytics

This article describes an architecture for many models that uses Azure Machine Learning and compute clusters. A many-models architecture provides versatility for situations that require complex setup.

Architecture



Download a [Visio file](#) of this architecture.

Dataflow

The following dataflow corresponds to the previous diagram:

1. Data ingestion:

- Azure Data Factory retrieves data from a source database and copies it to Azure Data Lake Storage.
- The data is then stored in a Machine Learning data store as a tabular dataset.

2. Model-training pipeline:

a. Prepare data:

- The training pipeline retrieves the data from the data store and transforms it as needed.
- The data is grouped into datasets for training the models.

b. Train models:

- i. The pipeline trains models for all the datasets created during data preparation.
- ii. It uses the `ParallelRunStep` class to train multiple models in parallel.
- iii. After the models are trained, the pipeline registers the models and their testing metrics in Machine Learning.

3. Model-promotion pipeline:

a. Evaluate models:

- i. The promotion pipeline evaluates the trained models before it moves them to production.
- ii. A DevOps pipeline applies business logic to determine whether a model meets the criteria for deployment. For example, it might verify that the accuracy on testing data exceeds 80%.

b. Register models:

- i. The promotion pipeline registers qualifying models into the production Machine Learning workspace.

4. Model batch-scoring pipeline:

a. Prepare data:

- i. The batch-scoring pipeline retrieves data from the data store and transforms each file as needed.
- ii. The data is grouped into datasets for scoring.

b. Score models:

- i. The pipeline uses the `ParallelRunStep` class to score multiple datasets in parallel.
- ii. It identifies the appropriate model for each dataset in Machine Learning by searching model tags.
- iii. The model is downloaded and used to score the dataset.
- iv. The `DataTransferStep` class writes the results back to Azure Data Lake.
- v. Predictions are passed from Azure Data Lake to Synapse SQL for serving.
- vi. The managed online endpoint provides real-time scoring.

vii. Because of the large number of models, they're loaded on demand instead of pre-loaded.

5. Results:

- **Predictions:** The batch-scoring pipeline saves predictions to Synapse SQL.
- **Metrics:** Microsoft Power BI connects to the model predictions to retrieve and aggregate results for presentation.

Components

- [Azure Data Factory](#) is a cloud-based data integration service that allows the creation of data-driven workflows for orchestrating and automating data movement and transformation. In this architecture, Azure Data Factory ingests enterprise data and third-party metadata into Data Lake Storage.
- [Azure DevOps](#) is a set of developer services that provide comprehensive application and infrastructure lifecycle management. It includes tools for continuous integration and continuous delivery (CI/CD) pipelines, work tracking, source control, build pipelines, package management, and testing solutions. In this architecture, Azure DevOps is used to manage CI/CD pipelines for automating model promotion, testing, and deployment to production environments.
- [Azure SQL Database](#) is a fully managed relational cloud database. In this architecture, SQL Database is used to store structured data that might be queried or analyzed as part of the data pipeline.
- [Azure Stream Analytics](#) is a real-time analytics and complex event-processing service designed to analyze and process high volumes of fast streaming data. In this architecture, Stream Analytics can be used for real-time data processing.
- [Azure Synapse Analytics](#) is an analytics service that unifies data integration, enterprise data warehousing, and big data analytics. It's used in this architecture to store batch-scoring results. This approach enables efficient querying and retrieval of predictions for reporting or analysis. Synapse SQL is used to serve predictions to downstream applications and enable visualization tools like Power BI to access aggregated results.
- [Data Lake Storage](#) is a massively scalable and secure storage service for high-performance analytics workloads. In this architecture, Data Lake Storage serves as the primary storage layer for raw and transformed datasets, and for storing results from scoring pipelines.

- **Machine Learning** is an enterprise-grade machine learning service for building and deploying models quickly. It provides users at all skill levels with tools such as a low-code designer, automated machine learning, and a hosted Jupyter notebook environment that supports various integrated development environments. In this architecture, Machine Learning is used to manage the lifecycle of models, including training, evaluation, and deployment. It also orchestrates pipelines for tasks such as training, promotion, and scoring.
 - **Managed online endpoints** are a feature of Machine Learning used for real-time scoring. In this architecture, a managed online endpoint helps provide a scalable and secure way to serve predictions in near real-time by loading machine learning models on demand.
 - The **ParallelRunStep class** is a component of Machine Learning pipelines used to run parallel jobs efficiently. It enables scalable processing of batch tasks, such as training or scoring many models simultaneously. In this architecture, the `ParallelRunStep` class is used in both the model-training and batch-scoring pipelines to train or score multiple datasets or models in parallel, which significantly reduces the runtime of these operations.
- **Power BI** is a collection of software services, apps, and connectors that work together to turn unrelated sources of data into coherent, visually immersive, and interactive insights. In this architecture, Power BI connects to Synapse SQL to retrieve and present predictions and aggregated metrics through interactive dashboards.

Alternatives

- You can use any database for source data.
- You can use Azure Kubernetes Service (AKS) for real-time inferencing instead of managed online endpoints. AKS allows you to deploy containerized models and provides more control over deployment. These capabilities enable dynamic loading of models to handle incoming requests without depleting resources.

Scenario details

Many machine learning problems are too complex for a single machine learning model to solve. Whether it's predicting sales for every item of every store or modeling maintenance for hundreds of oil wells, having a model for each instance might improve results on many machine learning problems. This *many models* pattern is common across a wide range of industries, and has many real-world use cases. With the use of Machine Learning, an end-to-

end many models pipeline can include model training, batch-inferencing deployment, and real-time deployment.

A many models solution requires a different dataset for every model during training and scoring. For instance, if the task is to predict sales for each item in every store, each dataset corresponds to a unique item-store combination.

Potential use cases

- **Retail:** A grocery store chain needs to create a separate revenue forecast model for each store and item, totaling over 1,000 models for each store.
- **Supply chain:** For each combination of warehouse and product, a distribution company needs to optimize inventory.
- **Restaurants:** A chain with thousands of franchises needs to forecast the demand for each franchise.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Well-Architected Framework](#).

- **Data partitions:** Dividing the data into partitions is essential for implementing the many models pattern. If you want one model for each store, each dataset contains all the data for a single store, so there are as many datasets as there are stores. If you want to model products by store, there's a dataset for every combination of product and store. Depending on the source data format, it might be easy to partition the data, or it might require extensive data shuffling and transformation. Spark and Synapse SQL scale well for these tasks, while Python pandas doesn't because it runs on a single node and process.
- **Model management:** The training and scoring pipelines identify and invoke the right model for each dataset. They do this by calculating tags that characterize the dataset, and then use the tags to find the matching model. The tags identify the data partition key and the model version, and might also provide other information.
- **Choose the right architecture:**
 - Spark is suitable when your training pipeline has complex data transformation and grouping requirements. It provides flexible splitting and grouping techniques to group data by combinations of characteristics, such as product-store or location-product. The results can be placed in a Spark DataFrame for use in subsequent steps.

- If your machine learning training and scoring algorithms are straightforward, you might be able to partition data with libraries such as scikit-learn. In this scenario, you might not need Spark, so you can avoid possible complexities that arise when you install Azure Synapse Analytics or Azure Databricks.
 - If your training datasets are already created, like when they're stored in separate files or organized into distinct rows or columns, you don't need Spark for complex data transformations.
 - The Machine Learning and compute clusters solution provides versatility for situations that require complex setup. For example, you can make use of a custom Docker container, download files, or download pretrained models. Computer vision and natural language processing deep learning are examples of applications that might require this versatility.
- **Separate model repos:** To protect the deployed models, consider storing them in their own repository that the training and testing pipelines don't access.
 - **ParallelRunStep class:** The Python [ParallelRunStep class](#) is a powerful option for running many models training and inferencing. It can partition your data in various ways and then apply your machine learning script on elements of the partition in parallel. Like other forms of Machine Learning training, you can specify a custom training environment that has access to Python Package Index (PyPI) packages, or a more advanced custom Docker environment for configurations that require more than standard PyPI. There are many CPUs and GPUs to choose from.
 - **Online inferencing:** If a pipeline loads and caches all models from the beginning, the models might deplete the container's memory. Therefore, load the models on demand in the run method, even though it might increase latency slightly.

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

To better understand the cost to run this scenario on Azure, use the [pricing calculator](#). You should assume that:

- The serving models are trained daily to keep them current.
- You need about 30 minutes to process a dataset that contains 40 million rows of 10 thousand unique store-product combinations. The dataset trains on Azure Databricks by

using a cluster that's provisioned with 12 virtual machines (VMs) that use Ls16_v2 instances. Batch scoring with the same set of data takes about 20 minutes.

- You can use Machine Learning to deploy real-time inferencing. Depending on your request volume, choose a suitable type of VM and cluster size.
- An AKS cluster automatically scales as needed, which results in an average of two active nodes each month.

To see how pricing differs for your use case, change the variables in the pricing calculator to match your expected data size and serving load requirements. For larger or smaller training data sizes, increase or decrease the size of the Azure Databricks cluster. To handle more concurrent users during model serving, increase the AKS cluster size.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal author:

- [James Nguyen](#) | Principal Cloud Solution Architect

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Configure a Kubernetes cluster for Machine Learning](#)
- [ParallelRunStep class](#)
- [DataTransferStep class](#)
- [Create datastores](#)
- [What is Azure Synapse Analytics?](#)
- [Deploy a model to an AKS cluster](#)

Related resources

- [Analytics architecture design](#)
- [Choose an analytical data store in Azure](#)
- [Choose a data analytics technology in Azure](#)

Extract and map information from unstructured content

Azure AI services

Azure Cosmos DB

Azure Container Apps

Azure AI Foundry

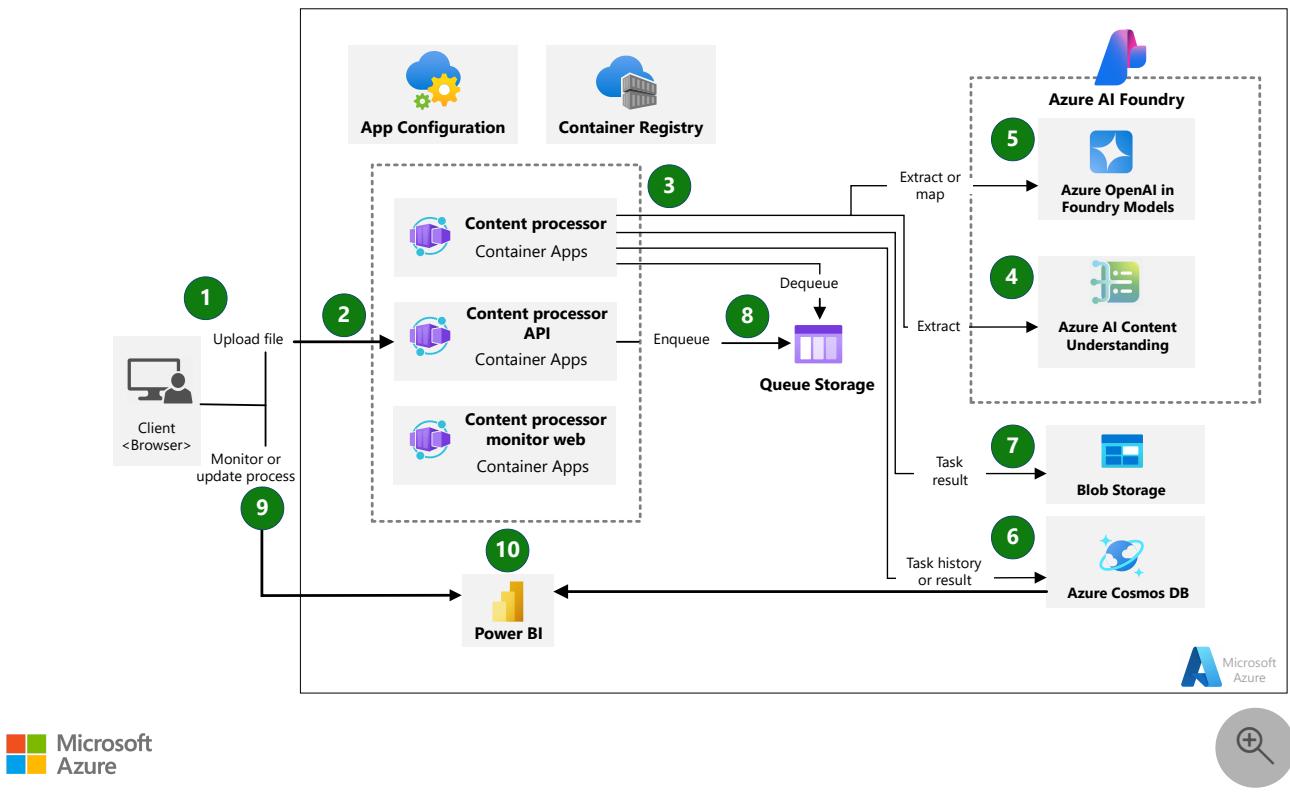
Solution ideas

This article describes a solution idea. Your cloud architect can use this guidance to help visualize the major components for a typical implementation of this architecture. Use this article as a starting point to design a well-architected solution that aligns with your workload's specific requirements.

This architecture describes a content processing solution that extracts data and applies schemas across multi-modal content by using confidence scoring and user validation. It processes claims, invoices, contracts, and other documents by extracting information from unstructured content and mapping it to structured formats. This architecture applies Azure AI Foundry, Azure AI Content Understanding, Azure OpenAI in Azure AI Foundry Models, and other Azure services to transform large volumes of unstructured content through event-driven processing pipelines.

This architecture shows how to build scalable systems for processing content. The systems handle text, images, tables, and graphs and include automatic quality checks and human review for business document workflows.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

The following workflow corresponds to the previous diagram:

1. Users upload multi-modal content, like documents, images, contracts, and invoices, through the web front-end interface. Content is submitted with specific processing requirements and target schemas.
2. The Azure Container Apps website receives the content upload request and invokes the processing API hosted in Container Apps. Both components are custom-coded solutions tailored for this scenario. The API selects the appropriate processing pipeline and initiates content analysis workflows.
3. Container Apps manages the processing workflow. It connects Content Understanding, which performs optical character recognition (OCR) and extracts text, with Azure OpenAI in Foundry Models. These models map schemas and convert the extracted data into structured formats.
4. Content Understanding performs machine learning-based OCR for efficient text extraction from various content formats, including images, tables, and graphs.
5. Azure OpenAI in Foundry Models with GPT Vision processes the extracted content, maps it to custom or industry-defined schemas, and generates a structured JSON output with

confidence scoring.

6. The orchestration code in Container Apps stores processed results, confidence scores, schema mappings, and historical processing data for audit trails and continuous improvement in Azure Cosmos DB.
7. The orchestration code in Container Apps uses Azure Blob Storage to store source documents, intermediate processing artifacts, and final structured outputs for reliable data persistence and retrieval.
8. Azure Queue Storage manages event-driven processing workflows between this solution's services. This management ensures reliable message handling and processing coordination across the pipeline components.
9. The content processor monitor website displays the processed results to users through the web interface. Users can review the structured JSON output, correct any inaccuracies, add comments for context or feedback, and save the final validated results to the system.
10. The content processor monitor website feeds processing metrics and user feedback data directly into Power BI dashboards. Processed data and metadata stored in Azure Cosmos DB provide comprehensive analytics on the content processing pipeline. These insights include KPIs, success rates, document type distributions, confidence score trends, user correction patterns, and other operational metrics that support data-driven optimization of the content processing pipeline.

Components

- [Container Apps](#) is a serverless container platform that you can use to run microservices and containerized applications on a serverless platform. In this architecture, Container Apps hosts the processing pipeline API that orchestrates content analysis, coordinates between AI services, and manages the extraction and transformation workflows. The code that runs is custom coded by your software engineering team.
- [Azure AI Foundry](#) is a managed AI service that provides access to advanced language models for natural language processing and generation. In this architecture, Azure AI Foundry provides the foundation for deploying and managing AI models used in the content processing pipeline and is the gateway into the connected AI services, like Content Understanding.
 - [Azure OpenAI in Foundry Models](#) is a component of Azure AI Foundry that provides language models, including GPT-4o and GPT-4o mini. In this architecture, the models are hosted as a service in Azure AI Foundry. These models perform schema-based data

transformation, map extracted content to structured formats, and calculate confidence scores for extraction accuracy.

- [Content Understanding](#) is a multi-modal AI service that analyzes various media content, such as audio, video, text, and images, and transforms it into structured, searchable data. In this architecture, Content Understanding accurately performs advanced OCR and content extraction from multi-modal documents.
- [Azure Cosmos DB](#) is a globally distributed, multiple-model database service that provides guaranteed low latency and elastic scalability. In this architecture, Azure Cosmos DB stores processed results, confidence scores, validation outcomes, and historical processing data for audit trails and performance optimization.
- [Blob Storage](#) is Microsoft's object storage solution optimized for storing massive amounts of unstructured data. In this architecture, Blob Storage maintains source documents, intermediate processing artifacts, and final structured outputs with reliable durability and global accessibility.
- [Azure Container Registry](#) is a managed Docker registry service that stores and manages container images. In this architecture, Container Registry manages versioned container images for the processing pipeline components. This system ensures consistent deployment and rollback capabilities.
- [Power BI](#) is a collection of software services, apps, and connectors that work together to help you create, share, and consume business insights in the way that best serves you and your organization. In this architecture, Power BI connects to Azure Cosmos DB and receives real-time processing metrics from the monitoring web application to deliver comprehensive analytics on document processing performance, user feedback patterns, and operational KPIs.

Scenario details

This content processing solution addresses the challenge of extracting meaningful data from large volumes of unstructured, multi-modal content that organizations receive daily. Traditional manual processing of documents such as contracts, invoices, claims, and compliance reports is time-consuming, error-prone, and doesn't scale with business growth. As a result, organizations face inconsistent data quality, lack of standardization, and difficulty integrating extracted information into downstream business processes.

This solution uses advanced AI services to automatically extract, transform, and validate content from various document types. The system provides confidence scoring to enable automated processing for high-confidence extractions while flagging lower-confidence results

for human review. This approach ensures both speed and accuracy while maintaining the flexibility to handle diverse content formats and custom business schemas.

Potential use cases

Financial services processing

Claims processing automation: Extract policy details, damage assessments, and cost estimates from insurance claims documents, photos, and adjuster reports by using automated validation and compliance checks.

Invoice and contract processing: Automatically extract vendor information, line items, terms, and conditions from invoices and contracts, and map them to enterprise systems by using confidence scoring for approval workflows.

Regulatory document analysis: Process regulatory filings, compliance reports, and audit documentation to extract key metrics and ensure adherence to financial regulations and reporting requirements.

Healthcare documentation

Clinical document processing: Extract patient information, diagnoses, treatment plans, and medication information from medical records, lab reports, and clinical notes for electronic health record integration.

Medical billing automation: Process medical claims, billing statements, and insurance forms to extract procedure codes, patient details, and coverage information for automated billing workflows.

Research data extraction: Analyze clinical trial documents, research papers, and patient consent forms to extract study parameters, outcomes, and compliance data for medical research workflows.

Legal and compliance

Contract analysis and extraction: Process legal contracts, agreements, and amendments to extract key terms, obligations, dates, and parties for contract management and compliance monitoring.

Legal document discovery: Analyze legal briefs, depositions, and case files to extract relevant facts, citations, and evidence for litigation support and case preparation.

Compliance documentation: Process regulatory submissions, audit reports, and compliance certificates to extract requirements, findings, and corrective actions for governance workflows.

Manufacturing and supply chain

Quality documentation processing: Extract inspection results, test data, and certification details from quality control documents and certificates for compliance tracking and process improvement.

Supplier documentation: Process vendor certifications, material specifications, and shipping documents to extract compliance data and supply chain information for procurement workflows.

Maintenance record analysis: Extract equipment data, maintenance schedules, and repair histories from technical documentation for predictive maintenance and asset management systems.

Alternatives

This architecture includes multiple components that you can substitute with other Azure services or approaches, depending on your workload's functional and nonfunctional requirements. Consider the following alternatives and trade-offs.

Content extraction approach

Current approach: This solution uses Content Understanding for advanced OCR and content extraction combined with Azure OpenAI for schema mapping and transformation. This approach provides high accuracy for complex multi-modal content with flexible schema customization.

Alternative approach: Use Azure AI Document Intelligence for document processing by using prebuilt models for common document types like invoices, receipts, and forms. This approach provides faster implementation for standard document types but less flexibility for custom schemas.

Consider this alternative if your workload has the following characteristics:

- You primarily process standard document types that have well-defined formats.
- You need faster time-to-market with prebuilt extraction models.
- Your schema requirements align with standard document intelligence models.

- You have limited custom development resources for schema mapping.

Processing orchestration

Current approach: This solution uses Container Apps to host custom processing logic that orchestrates the content analysis pipeline. This approach provides maximum control over processing workflows, error handling, and custom business logic integration.

Alternative approach: Use Azure Logic Apps or Azure Functions for workflow orchestration with built-in connectors to AI services. This approach provides visual workflow design and managed service benefits but less control over processing logic.

Consider this alternative if your workload has the following characteristics:

- You prefer visual workflow design over custom code development.
- Your processing workflows are relatively simple and use standard conditional logic.
- You want to minimize infrastructure management overhead.
- Your team has more expertise in low-code and no-code solutions than in containerized applications.

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

For more information about the costs to run this scenario, see the preconfigured [estimate in the Azure pricing calculator](#).

Pricing varies by region and usage, so it's not possible to predict exact costs for your deployment. Most Azure resources used in this infrastructure follow usage-based pricing tiers. However, Container Registry incurs a fixed cost per registry per day.

Deploy this scenario

To deploy an implementation of this architecture, follow the steps in the [GitHub repo](#).

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal author:

- [Solomon Pickett](#) | Software Engineer II

Other contributor:

- [Todd Herman](#) | Principal Software Engineer

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Content Understanding documentation](#)
- [Azure OpenAI documentation](#)
- [Content processing solution implementation deployment guide](#)

Generate documents from your data

Azure AI services

Azure Cosmos DB

Azure AI Foundry

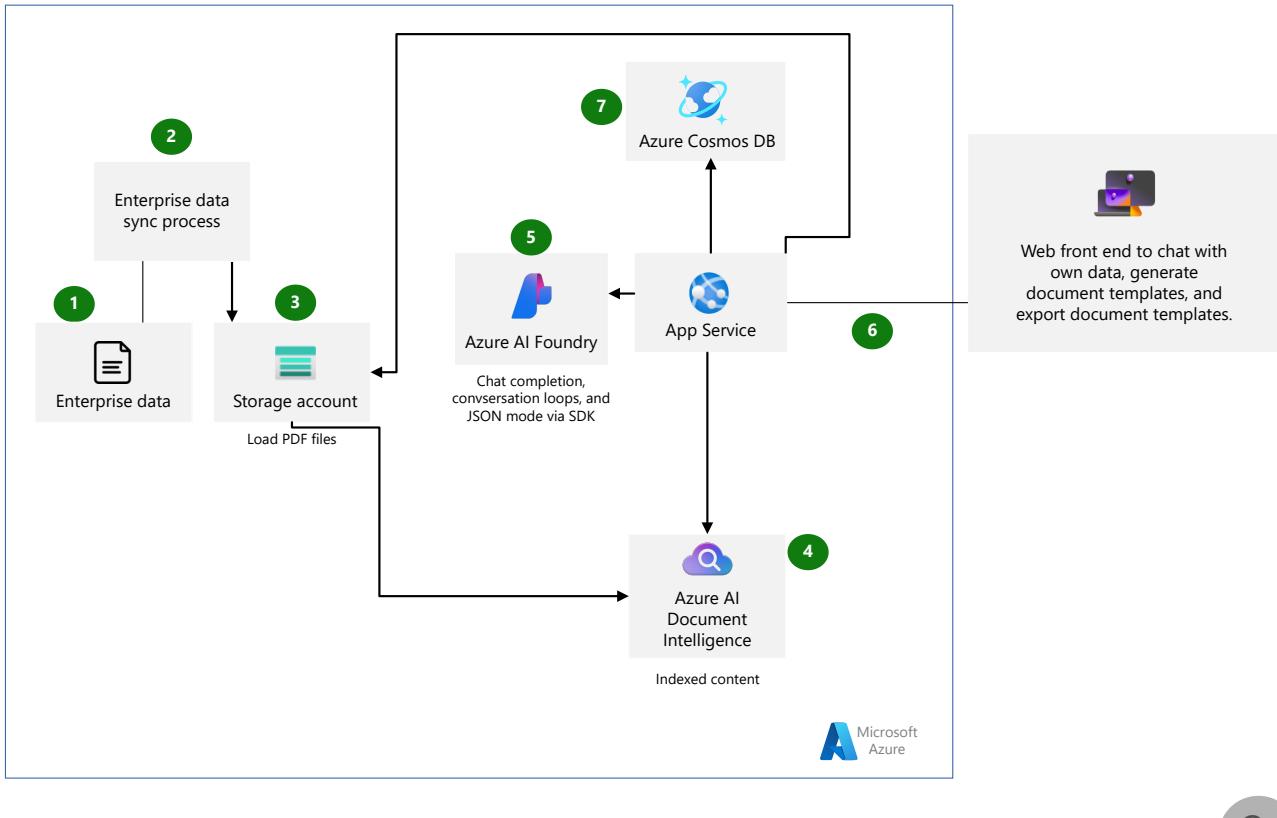
Solution ideas

This article describes a solution idea. Your cloud architect can use this guidance to help visualize the major components for a typical implementation of this architecture. Use this article as a starting point to design a well-architected solution that aligns with your workload's specific requirements.

This architecture demonstrates a document generation solution that enables organizations to create intelligent structured and unstructured documents grounded in their enterprise data. The solution uses Azure AI Document Intelligence to identify relevant data, summarize information, and generate contextual content through conversational interactions. Users can generate documents based on this organizational knowledge and receive them in Word format.

The architecture combines retrieval, summarization, and generation with document persistence to support faster document creation workflows. The system enables user interaction through natural language and helps embed organizational knowledge directly into document processing workflows. It also caches generated content to avoid regeneration overhead and accelerate document creation.

Architecture



Microsoft Azure



Download a [Visio file](#) of this architecture.

Workflow

The following workflow corresponds to the previous diagram:

1. Line-of-business applications or other processes in the organization generate enterprise documents and reference materials that serve as foundational knowledge for document generation.
2. A synchronization process periodically manages the ingestion and updating of enterprise data from various sources into this workload.
3. An Azure Storage account receives and stores enterprise documents, including PDF files. It makes them available for downstream services to process and index. A storage account also stores the generated documents from user sessions later.
4. Azure AI Document Intelligence creates searchable indexes from the processed and enriched documents, which enables semantic search capabilities and rapid information retrieval for document generation. Indexing skills might maintain the index in Azure AI Search.

5. Azure AI Foundry uses the indexed content to power conversational interactions through chat completion, conversation loops, and JSON mode via SDK. This process generates contextual documents based on user queries and organizational data.
6. Azure App Service hosts the web front end where users interact with the system by using natural language to generate documents.
7. Azure Cosmos DB stores conversation history and user interactions, while maintaining context for continuous improvement.

Components

- [App Service](#) is a platform as a service (PaaS) solution that provides a scalable web hosting environment for applications. In this architecture, App Service hosts the web front-end interface where users interact with their enterprise data through conversational AI functionality. App Service also generates DOCX files by using the docx React library and stores them in Storage for delivery. The interface enables both structured and unstructured document generation and DOCX export capabilities, which provides a responsive and intuitive user experience.
- [Azure AI Foundry](#) is a managed AI service that provides access to advanced language models for natural language processing and generation. In this architecture, Azure AI Foundry provides models as a service (MaaS) for the Semantic Kernel-based agents to invoke.
- [Azure AI Document Intelligence](#) is a cloud-based Azure AI service that uses machine learning models to automate data processing in applications and workflows. Azure AI Document Intelligence helps enhance data-driven strategies and enrich document search capabilities.
- [Azure Storage](#) is a Microsoft object storage solution optimized for storing massive amounts of unstructured data. In this architecture, a Storage account stores enterprise documents and reference materials, including PDF files, that provide the foundational knowledge base for the document generation process. A Storage account also stores generated documents for caching purposes.
- [Azure Cosmos DB](#) is a globally distributed, multi-model database service that provides guaranteed low latency and elastic scalability. In this architecture, Azure Cosmos DB stores conversation history and user interactions. This capability maintains context across sessions, enables intelligent document retrieval, and eliminates regeneration overhead for improved performance.

Scenario details

This document generation solution addresses the challenge that organizations encounter when they want to create consistent, high-quality business documents that use institutional knowledge. Traditional document creation often suffers from blank page syndrome, inconsistent formatting, missed relevant information, and significant time investment from subject matter experts. This solution transforms document creation through conversational AI that generates structured documents such as contracts, invoices, and promissory notes, and unstructured documents such as proposals, reports, and briefings, all grounded in organizational data.

This architecture supports transactional usage only. It enables focused, real-time document generation workflows that maintain quality and consistency for individual document requests. It doesn't support batch processing.

Potential use cases

Legal and compliance documentation

Contract template generation: Automatically generate contract templates based on previous agreements, legal precedents, and company policies. This approach ensures consistency and compliance across all business relationships.

Regulatory submission preparation: Create compliance documentation by synthesizing relevant regulations, organizational policies, and historical submission data into properly formatted regulatory filings.

Legal brief drafting: Generate legal document drafts by analyzing case law, precedents, and client information stored in an organization's knowledge base.

Business operations and proposals

Investment proposal creation: Synthesize market research, financial data, and strategic documents to generate comprehensive investment proposals tailored to specific opportunities and stakeholder requirements.

Grant application development: Create grant applications by combining project requirements, organizational capabilities, and historical successful submissions into compelling funding requests.

Requests for Proposals (RFP) response generation: Automatically draft responses to RFPs by analyzing requirements against organizational capabilities and previous successful proposals.

Financial and procurement documentation

Invoice template standardization: Generate consistent invoice templates that incorporate organizational branding, legal requirements, and customer-specific terms based on historical billing data.

Purchase order automation: Create purchase orders by referencing vendor databases, procurement policies, and budget constraints to ensure compliance and accuracy.

Financial report compilation: Generate financial reports by synthesizing data from multiple sources into standardized templates that meet regulatory and stakeholder requirements.

Healthcare and research applications

Clinical protocol documentation: Generate research protocols by combining regulatory requirements, institutional guidelines, and previous study designs into compliant and comprehensive documents.

Patient care plan templates: Create standardized care plan templates that incorporate best practices, institutional policies, and patient-specific considerations.

Research grant proposals: Develop research funding proposals by synthesizing scientific literature, institutional capabilities, and funding agency requirements.

Alternatives

This architecture includes multiple components that you can substitute with other Azure services or approaches, depending on your workload's functional and nonfunctional requirements. Consider the following alternatives and trade-offs.

Document generation approach

Current approach: Use custom AI-powered generation that includes enterprise data grounding and intelligent caching for both structured and unstructured documents.

Alternative approach: Use Azure AI Document Intelligence with prebuilt forms for structured documents only, combined with traditional document management systems.

Consider the alternative if your workload primarily focuses on standardized forms that have minimal unstructured content requirements.

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

This preconfigured [estimate in the Azure pricing calculator](#) shows the costs to run this scenario.

Pricing varies based on region and usage, so you can't predict exact costs for your scenario. Most of the Azure resources used in this infrastructure are on usage-based pricing tiers.

Deploy this scenario

To deploy an implementation of this architecture, follow the [deployment steps](#).

Next steps

- [AI Search documentation](#)
- Design and develop a retrieval-augmented generation (RAG) solution

Use AI enrichment with image and text processing

Azure App Service

Azure Blob Storage

Azure AI Search

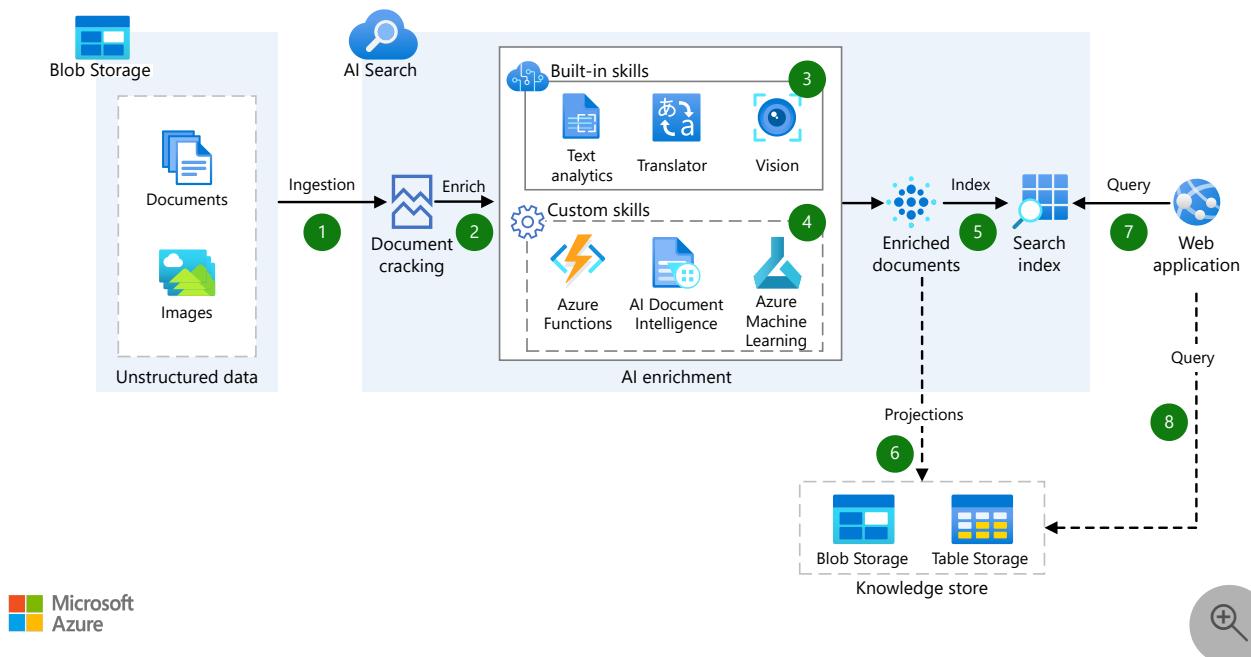
Azure Functions

Solution ideas

This article describes a solution idea. Your cloud architect can use this guidance to help visualize the major components for a typical implementation of this architecture. Use this article as a starting point to design a well-architected solution that aligns with your workload's specific requirements.

This article describes how to use image processing, natural language processing, and custom skills to capture domain-specific data. You can use that data to enrich text and image documents. Incorporate Azure AI Search with AI enrichment to help identify and explore relevant content at scale. This solution uses AI enrichment to extract meaning from the original complex, unstructured JFK Assassination Records (JFK Files) dataset.

Architecture



Download a [Visio file](#) of this architecture.

Dataflow

The following dataflow corresponds to the previous diagram. The dataflow describes how the unstructured JFK Files dataset passes through the AI Search skills pipeline to produce structured and indexable data.

1. Unstructured data in Azure Blob Storage, such as documents and images, is ingested into AI Search.
2. To initiate the indexing process, the *document cracking* step extracts images and text from the data and then enriches the content. The enrichment steps in this process depend on the data and type of skills that you select.
3. [Built-in skills](#) based on the Azure AI Vision and Azure AI Language APIs provide AI enrichments such as image optical character recognition (OCR), image analysis, text translation, entity recognition, and full-text search.
4. [Custom skills](#) support scenarios that require more complex AI models or services. Examples include Azure AI Document Intelligence, Azure Machine Learning models, and Azure Functions.
5. After the enrichment process is complete, the indexer saves the enriched and indexed documents in a [search index](#). Full-text search and other query forms can use this index.
6. The enriched documents can also project into a [knowledge store](#), which downstream apps like knowledge mining apps or data science apps can use.
7. Queries access the enriched content in the search index. The index supports custom analyzers, fuzzy search queries, filters, and a scoring profile to tune search relevance.
8. Applications that connect to Blob Storage or to Azure Table Storage can access the knowledge store.

Components

- [AI Search](#) is a search service that enables indexing, querying, and enrichment of content by using built-in and custom AI skills. You can use AI Search to apply [prebuilt AI skills](#) to content. In this architecture, it indexes the content and powers the search user experience. This architecture also uses the service's extensibility mechanism to add [custom skills](#), which provide specific enrichment transformations.
- [Azure AI Vision](#) is a service that extracts text and visual information from images. In this architecture, it uses [text recognition](#) to extract and recognize text information from images. The [Read API](#) uses OCR recognition models and is optimized for large, text-heavy documents and noisy images.

- [Azure AI Language](#) is a text analytics service that extracts structured information from unstructured text by using capabilities like [named entity recognition](#) and [key phrase extraction](#). In this architecture, Language enriches the JFK Files by identifying named entities and key phrases to support semantic search and filtering.
- [Blob Storage](#) is a REST-based object storage solution optimized for large volumes of unstructured data. You can use Blob Storage to expose data publicly or to store application data privately. In this architecture, Blob Storage stores the original JFK Files dataset, including scanned documents and images, which are ingested into the AI enrichment pipeline.
- [Table Storage](#) is a NoSQL storage service for structured and semi-structured data. In this architecture, Table Storage supports the knowledge store, which enables downstream applications to access enriched and indexed data.
- [Azure Functions](#) is a serverless compute service that runs small pieces of event-triggered code without having to explicitly provision or manage infrastructure. In this architecture, a Functions method applies the Central Intelligence Agency (CIA) cryptonyms list to the JFK Files as a custom skill.
- [Azure App Service](#) is a managed platform for building and hosting web applications. In this architecture, it hosts a standalone web app that demonstrates the enriched search experience and allows users to explore connections within the indexed JFK documents.

Scenario details

Large, unstructured datasets can include typewritten and handwritten notes, photos, diagrams, and other unstructured data that standard search solutions can't parse. The [JFK Files](#) contain over 34,000 pages of documents about the CIA investigation of the 1963 JFK assassination.

You can use AI enrichment in AI Search to extract and enhance searchable, indexable text from images, blobs, and other unstructured data sources like the JFK Files. AI enrichment uses pretrained machine learning skill sets from the Azure AI services [Vision](#) and [Language](#) APIs. You can also create and attach [custom skills](#) to add special processing for domain-specific data like CIA cryptonyms. AI Search can then index and search that context.

The AI Search skills in this solution can be categorized into the following groups:

- **Image processing:** This solution uses built-in [text extraction](#) and [image analysis](#) skills, including object and face detection, tag and caption generation, and celebrity and landmark identification. These skills create text representations of image content, which you can search by using the query capabilities of AI Search. *Document cracking* is the process of extracting or creating text content from nontext sources.

- **Natural language processing:** This solution uses built-in skills like [entity recognition](#), [language detection](#), and [key phrase extraction](#) that map unstructured text to searchable and filterable fields in an index.
- **Custom skills:** This solution uses custom skills that extend AI Search to apply specific enrichment transformations to content. You can [specify the interface for a custom skill](#) through the [custom web API skill](#).

Potential use cases

The JFK Files [sample project](#) and [online demo](#) presents a specific AI Search use case. This solution idea isn't intended to be a framework or scalable architecture for all scenarios. Instead, this solution idea provides a general guideline and example. The code project and demo create a public website and publicly readable storage container for extracted images, so you shouldn't use this solution with nonpublic data.

You can also use this architecture to perform the following actions:

- Increase the value and utility of unstructured text and image content in search apps and data science apps.
- Use custom skills to integrate open-source code, non-Microsoft code, or Microsoft code into indexing pipelines.
- Make scanned JPG, PNG, or bitmap documents full-text searchable.
- Produce better outcomes than standard PDF text extraction for PDFs with combined image and text. Some scanned and native PDF formats might not parse correctly in AI Search.
- Create new information from inherently meaningful raw content or context that's hidden in large, unstructured documents or semi-structured documents.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal author:

- [Carlos Alexandre Santos](#) | Senior Specialized AI Cloud Solution Architect

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

Learn more about this solution:

- [JFK Files project ↗](#)
- [Video: Use AI Search to understand the JFK documents](#)
- [JFK Files online demo ↗](#)

Read product documentation:

- [AI enrichment in AI Search](#)
- [What is Vision?](#)
- [What is Language?](#)
- [What is OCR?](#)
- [What is named entity recognition in Language?](#)
- [Introduction to Blob Storage](#)
- [Introduction to Functions](#)

Try the learning path:

- [Implement knowledge mining with AI Search](#)

Extract text from objects using Power Automate and AI Builder

AI Builder

Azure AI Document Intelligence

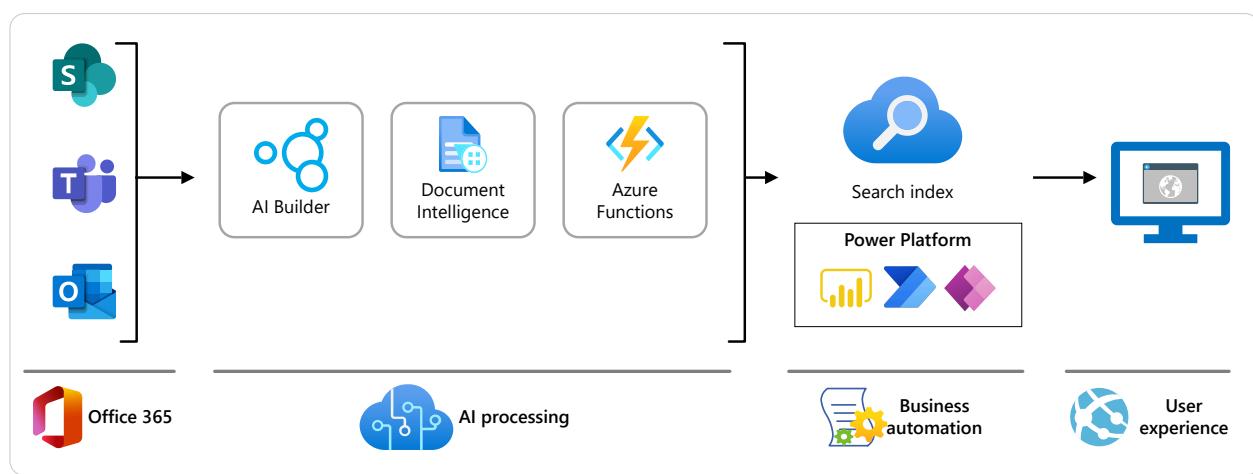
Power Automate

Microsoft Power Platform

Azure Functions

This article presents a solution for extracting text from images so it can be indexed and retrieved in SharePoint. By using AI Builder and Azure AI Document Intelligence, you can configure a Power Automate workflow to use a trained model to extract text from an image. Once you've configured a workflow, you can quickly search documents for meaningful text that's embedded in shapes and objects.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

1. An object detection model is trained in AI Builder to recognize objects that a user specifies.
2. A new document enters a SharePoint document library, OneDrive, or Teams.
3. The document's arrival triggers a Power Automate event. That event:
 - a. Runs the AI Builder model. AI Builder returns a JSON file that contains the pixel coordinates of any specified objects.
 - b. Sends the document to Document Intelligence for a full optical character recognition (OCR) scan. Document Intelligence returns a JSON file that contains scanned-in text and pixel coordinates of the text.

- c. Runs a function in Azure Functions. The function analyzes the pixel coordinates in the AI Builder and Document Intelligence output files. If detected objects intersect with scanned-in text, the function returns the matched data in a JSON file.
 - d. Enters the metadata, or the text from detected objects, into a document library.
4. The metadata is captured in a SharePoint search index.
5. Users search for the metadata by using PnP Modern Search web parts.

Components

- [AI Builder](#) is a Microsoft Power Platform capability. Use AI Builder to train models to recognize objects in images. AI Builder also offers prebuilt models for object detection.
- [Document Intelligence Studio](#) uses machine-learning models to extract and analyze form fields, text, and tables from your documents.

ⓘ Note

Before using Document Intelligence Studio, evaluate if your scenario works from within Azure AI Foundry. The capabilities found in the Document Intelligence Studio are being migrated to Azure AI Foundry. To help you select a portal experience, see [Choose the correct studio experience](#).

- [Power Automate](#) is a part of Microsoft Power Platform no-code or low-code intuitive solutions. Power Automate is an online workflow service that automates actions across apps and services.
- [Azure Functions](#) is an event-driven serverless compute platform. Azure Functions runs on demand and at scale in the cloud.
- [PnP Modern Search](#) solution is a set of SharePoint in Microsoft 365 modern web parts. By using these tools, you can create highly flexible and personalized search-based experiences.

Alternatives

- [Azure AI Document Intelligence](#) can do a full OCR scan of documents, with the resulting metadata stored in SharePoint.
- SharePoint can run OCR scans on documents and add content output to the index for retrieval. Use search techniques to target key information in documents.
- If you want to process a high rate of documents, consider using Azure Logic Apps to configure the components. Azure Logic Apps helps avoid consumption limits primarily

through its integration with a dedicated workflow engine and its ability to run under different pricing tiers, and is cost-effective. For more information, see [Azure Logic Apps](#).

Scenario details

Schematic and industrial diagrams often have objects that contain text. Manually scanning documents for relevant text can be laborious and time consuming.

Potential use cases

Use cases include:

- Complicated engineering schematic diagrams that contain various types of objects. By using this solution, you can quickly search for specific components on a diagram. Having access to embedded text in objects is helpful for investigations, exposing shortages, or looking for recall and failure notices.
- Industrial diagrams that show the components in a manufacturing assembly. This solution promptly identifies pumps, valves, automated switches, and other components. Identifying components helps with preventative maintenance, isolating hazardous components, and increasing the visibility of risk management in your organization.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Steve Pucelik](#) | Sr. Product Manager

Next steps

- Understand the types of documents that would be well suited for this solution. Typical documents include schematic diagrams, manufacturing control processes, and diagrams that contain many shapes that need to be isolated. For more information, see [Document Intelligence custom models](#).
- Become familiar with the capabilities that AI Builder offers. For more information, see [AI Builder in Power Automate overview](#).
- Define an information architecture that can receive and process your metadata. For more information, see [Azure AI Search skill set](#).
- For more information about how the solution works and whether it's suitable for your use cases, see [Extract text from objects](#).

Build a multiple-agent workflow automation solution with Microsoft Agent Framework

Azure Container Apps

Azure AI services

Azure Cosmos DB

Semantic Kernel

Azure AI Foundry

Solution ideas

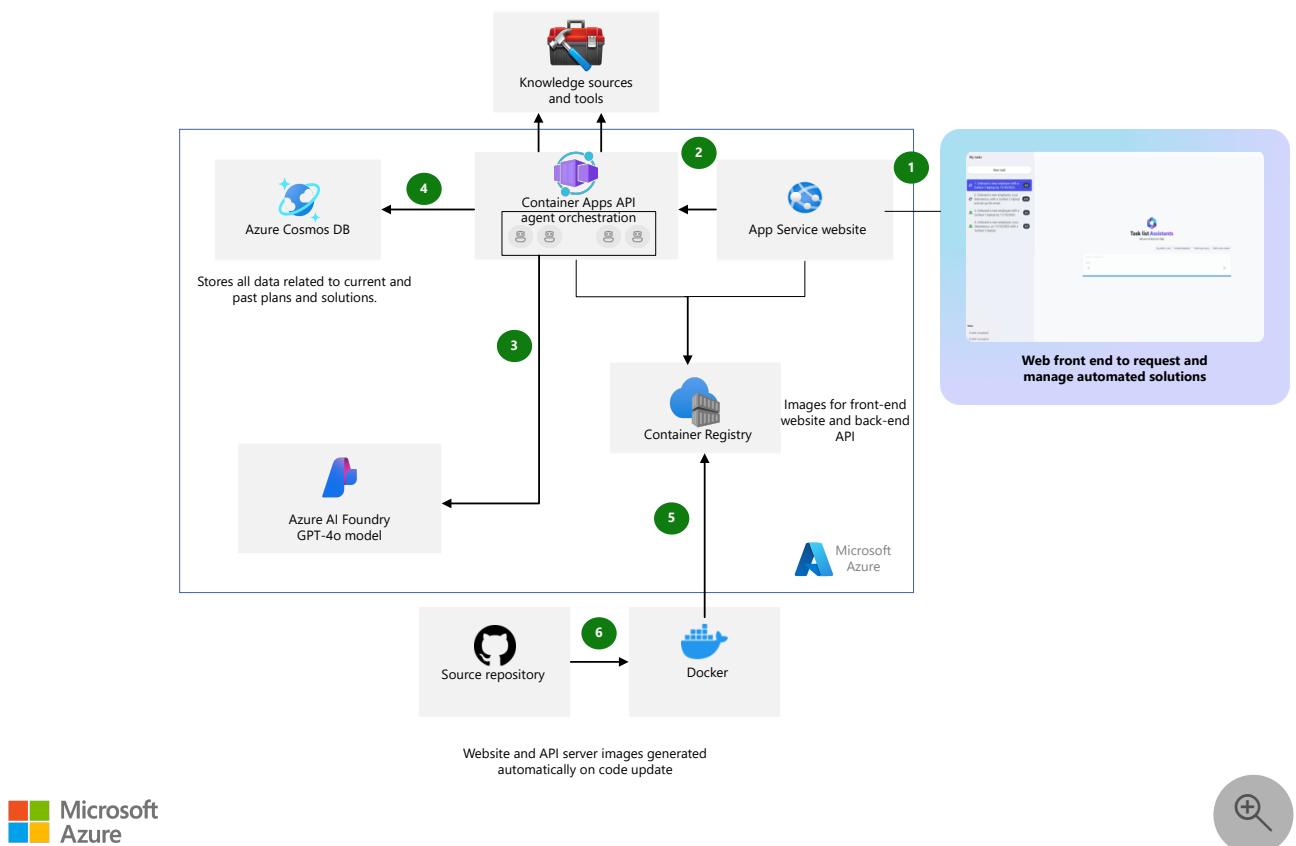
This article describes a solution idea. Your cloud architect can use this guidance to help visualize the major components for a typical implementation of this architecture. Use this article as a starting point to design a well-architected solution that aligns with your workload's specific requirements.

This architecture shows a process automation system that uses multiple AI agents. The agents are deployed into Azure Container Apps and use Azure AI services. This architecture's agents and orchestration behavior are defined in custom software with Microsoft Agent Framework. The architecture hosts specialized multiple AI agents that coordinate and run organizational tasks automatically.

This article highlights the infrastructure and DevOps aspects of how to manage multiple-agent systems on Azure, including continuous integration, data persistence, and agent coordination.

The architecture describes how to build scalable automation pipelines in which multiple AI agents collaborate through a central API orchestrator. It supports persistent learning and automated deployment processes for enterprise-grade task automation.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

The following workflow corresponds to the previous diagram:

1. Employees access the web front end to request and manage automated solutions. Tasks are submitted through the web interface with specific requirements and parameters.
2. The Azure App Service website receives the user request from the front end and calls an API hosted in Container Apps. That API processes the incoming task and determines which specialized AI agents are needed. The task is broken down into component parts for multiple-agent coordination.
3. The Container Apps API connects to an Azure AI Foundry-hosted GPT-4o model. Multiple specialized AI agents are orchestrated to handle different aspects of the task. Agents collaborate to plan, perform, and validate the tasks to be done.
4. Azure Cosmos DB stores all data related to current and past plans and solutions. Historical task data and patterns are maintained for learning and optimization purposes. Agent decisions and outcomes are persisted for future reference.
5. Azure Container Registry manages images for the front-end website and back-end API. This registry also maintains versioned container images for rollback capabilities.

6. The GitHub source repository triggers automatic builds of website and API server images on code updates. Docker then builds and deploys the updated container images to the registry.

Components

- [App Service](#) is a platform as a service solution that provides a scalable web hosting environment for applications. In this architecture, the App Service website serves as the front-end interface for users to request and manage automated solutions. It provides a responsive web experience for submitting tasks and tracking their progress.
- [Container Apps](#) is a serverless container platform that enables you to run microservices and containerized applications on a serverless platform. In this architecture, the Container Apps API serves as the central orchestration layer that processes user requests, coordinates multiple AI agents, and manages the completion state of tasks. It hosts the custom-developed code created by your software team that uses Microsoft Agent Framework.
- [Azure AI Foundry](#) is a managed AI service that provides access to advanced language models for natural language processing and generation. In this architecture, Azure AI Foundry provides models as a service for the Microsoft Agent Framework-based agents to invoke.
- [Azure Cosmos DB](#) is a globally distributed, multiple-model database service that provides low latency and elastic scalability. In this architecture, Azure Cosmos DB stores all data related to current and past automation plans and solutions. The Container Apps API writes data when new plans are created or tasks are run. The API reads data when users access their automation history via the App Service website.
- [Container Registry](#) is a managed Docker registry service that stores and manages container images. In this architecture, Container Registry manages images for both the front-end website and back-end API. This setup ensures consistent deployment and version control of the multiple-agent system components across environments.

Scenario details

This custom, multiple-agent automation engine addresses the challenge of coordinating complex, cross-departmental business processes that traditionally required significant manual oversight and coordination. Organizations often struggle with tasks that span multiple areas of expertise, demand consistent performance across teams, and require audit trails to support compliance.

This solution uses custom-coded, specialized AI agents that collaborate to break down complex organizational tasks into manageable components. Each agent contributes domain-specific knowledge and capabilities. This approach enables the system to manage sophisticated workflows that would otherwise require human coordination across multiple departments. The architecture scales through containerized deployment, preserves learning via persistent data storage, and supports continuous improvement through automated integration and delivery pipelines.

Other applications of this architecture are in code modernization and legacy system migration, where organizations face the dual challenge of technical complexity and business continuity requirements. Legacy systems often lack proper documentation, contain outdated programming languages, and embed critical business logic that must be preserved during migration. The multiple-agent approach works by coordinating specialized expertise across technical translation, business analysis, quality assurance, and documentation generation.

Potential use cases

Consider the following potential use cases for multiple-agent workflow automation.

Code modernization and migration

Legacy SQL query translation: Coordinate multiple specialized agents to translate SQL queries across different database dialects while you preserve business logic and performance characteristics. A SQL analysis agent identifies dialect-specific constructs. A translation agent converts syntax to the target platform. A validation agent tests query equivalence. A documentation agent generates migration notes. This approach addresses the common challenge of maintaining functional equivalence when you migrate from platforms like Oracle to Azure SQL Database or PostgreSQL.

Legacy application modernization: Orchestrate agents that specialize in code analysis, business logic extraction, architecture assessment, and modernization planning. Agents collaborate to analyze legacy codebases, extract embedded business rules, assess technical debt, generate modernization roadmaps, and create comprehensive documentation that captures institutional knowledge often lost during transitions.

Database schema migration: Coordinate agents for schema analysis, data type mapping, constraint translation, and validation testing. The multiple-agent system ensures that complex database structures, relationships, and business rules are accurately translated while it maintains data integrity and performance requirements.

Enterprise process automation

Employee onboarding orchestration: Coordinate IT provisioning, HR documentation, facility access, training schedules, and compliance requirements across multiple departments.

Contract management workflow: Automate legal review, procurement approval, financial analysis, and vendor communication for complex business agreements.

Incident response coordination: Orchestrate technical remediation, stakeholder communication, documentation, and post-incident analysis across IT, security, and business teams.

Financial services and compliance

Regulatory compliance automation: Coordinate data collection, analysis, reporting, and submission across multiple regulatory frameworks simultaneously.

Loan processing pipeline: Automate credit analysis, risk assessment, documentation review, and approval workflows that include multiple specialist teams.

Audit preparation management: Coordinate evidence gathering, documentation preparation, stakeholder interviews, and compliance verification across business units.

Healthcare and research

Clinical trial management: Orchestrate patient recruitment, regulatory compliance, data collection, safety monitoring, and reporting across research teams.

Patient care coordination: Automate scheduling, treatment planning, insurance verification, and care team communication for complex medical cases.

Medical equipment procurement: Coordinate clinical requirements, technical specifications, vendor evaluation, and regulatory approval processes.

Manufacturing and supply chain

Product launch coordination: Orchestrate design finalization, manufacturing setup, quality assurance, marketing preparation, and distribution planning.

Supplier onboarding process: Automate qualification assessments, contract negotiations, system integrations, and performance monitoring setup.

Quality incident management: Coordinate investigation, root cause analysis, corrective actions, and supplier communication for problems with quality.

Alternatives

This architecture includes multiple components that you can substitute with other Azure services or approaches, depending on your workload's functional and nonfunctional requirements. Consider the following alternatives and trade-offs.

Agent orchestration

Current approach: This solution uses custom agent code, written with the Microsoft Agent Framework SDK, to orchestrate agents and their interactions. Container Apps serves as the central orchestrator compute that runs the code. The code coordinates the multiple AI agents that operate on active workflows. This approach is a code-first solution that provides maximum control over agent behavior, orchestration logic, and compute scale.

Alternative approach: Use Azure AI Foundry Agent Service to define agents and connect them individually to relevant knowledge stores and tools. This approach is a no-code solution where you define agent behavior and agent relationships through a system prompt. The agents are hosted on your behalf, and you have no control over the compute that runs the agents.

Consider this alternative if your workload has the following characteristics:

- You don't require deterministic agent orchestration. You can sufficiently define agent behavior, including knowledge store access and tool use, through a system prompt.
- You don't require full control of your agents' compute.
- You only need HTTPS-accessible tools, and your knowledge stores are compatible with Foundry Agent Service.

For organizations that have mixed requirements, a hybrid approach can be effective. Standard workflows use Foundry Agent Service while critical or highly customized processes use self-hosted orchestration on Container Apps.

Multi-agent orchestration patterns

When you design multi-agent automation systems, consider how agents must coordinate to accomplish complex workflows. This architecture uses a custom orchestrator that manages agent interactions, but the coordination patterns that you choose significantly affect system performance and reliability. Sequential patterns suit dependent tasks like document approval workflows. Concurrent patterns suit independent operations like data collection from multiple sources. Group chat patterns enable collaborative problem-solving. Handoff patterns give specialized agents the ability to handle different workflow phases. For detailed guidance about

how to implement these coordination strategies, see [AI agent orchestration patterns](#). This article provides architectural patterns and implementation considerations for various multi-agent scenarios.

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

For more information about the costs of running this scenario, see the preconfigured estimate in the [Azure pricing calculator](#).

Pricing varies by region and usage, so it's not possible to predict exact costs in advance. Most Azure resources in this infrastructure follow usage-based pricing models. But Container Registry incurs a daily fixed cost for each registry.

Deploy this scenario

To deploy an implementation of this architecture, follow the steps in the [GitHub repo](#).

Code modernization implementation

For specific implementation of the multi-agent workflows that perform SQL query modernization, see [Modernize your code implementation](#). It demonstrates how multiple AI agents coordinate to translate SQL queries between different database dialects. It also generates documentation and validation reports throughout the process.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal author:

- [Solomon Pickett](#) | Software Engineer II

Other contributor:

- [Mark Taylor](#) | Principal Software Engineer

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Overview of the Agent architecture](#) that uses Microsoft Agent Framework
- [Foundry Agent Service documentation](#)

Basic Microsoft Foundry chat reference architecture

Azure App Service

Azure Monitor

Azure AI Foundry

Azure AI Foundry Agent Service

Azure AI Foundry SDK

This article provides a basic architecture to help you learn how to run chat applications by using [Microsoft Foundry](#) and [Azure OpenAI in Foundry Models](#). The architecture includes a client user interface (UI) that runs in Azure App Service. To fetch grounding data for the language model, the UI uses an agent hosted in Foundry Agent Service to orchestrate the workflow from incoming prompts to data stores. The architecture runs in a single region.

Important

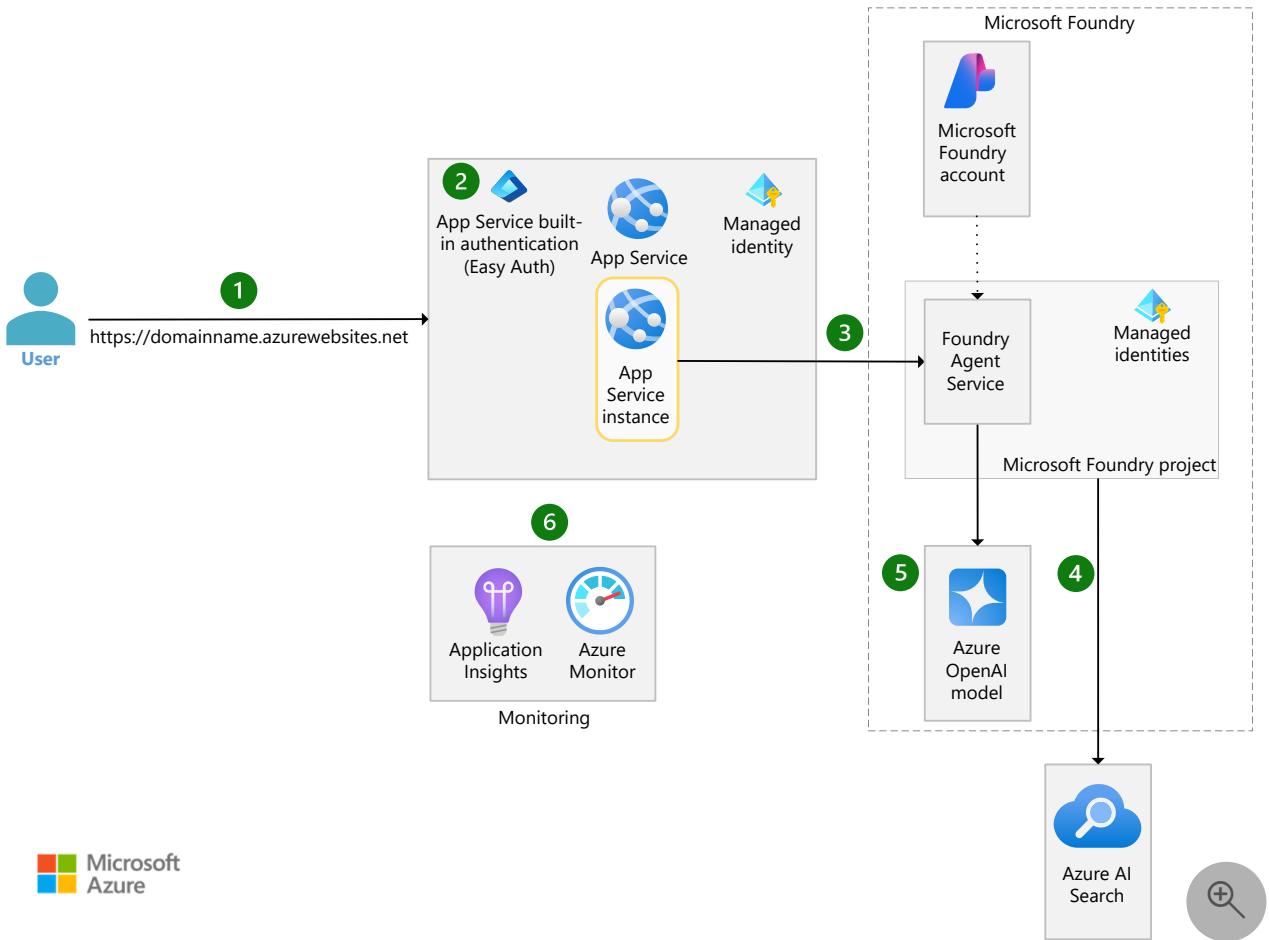
This architecture isn't for production. It's an introductory architecture for learning and proof of concept (POC) purposes. When you design production chat applications, use the [Baseline Foundry chat reference architecture](#), which adds production design decisions.

Important



An [example implementation](#) supports this guidance. It includes deployment steps for a basic end-to-end chat implementation. You can use this implementation as a foundation for your POC to work with chat applications that use Foundry agents.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

The following workflow corresponds to the previous diagram:

1. An application user interacts with a web application that contains chat functionality. They issue an HTTPS request to the App Service default domain on `azurewebsites.net`. This domain automatically points to the App Service built-in public IP address. The Transport Layer Security connection is established from the client directly to App Service. Azure fully manages the certificate.
2. The App Service feature called Easy Auth ensures that the user who accesses the website is authenticated via Microsoft Entra ID.
3. The application code deployed to App Service handles the request and renders a chat UI for the application user. The chat UI code connects to APIs hosted in the same App Service instance. The API code connects to an agent in Agent Service by using the [Azure AI Persistent Agents SDK](#).

4. Agent Service connects to Azure AI Search or requests up-to-date public knowledge to fetch grounding data for the query. The grounding data is added to the prompt that's sent to the model in the next step.
5. Agent Service connects to an Azure OpenAI model that's deployed in Foundry and sends the prompt that includes relevant grounding data and chat context.
6. Application Insights logs information about the original request to App Service and agent interactions.

Components

Many of this architecture's components are the same as the [basic App Service web application architecture](#) because the chat UI is based on that architecture. This section highlights data services, components that you can use to build and orchestrate chat flows, and services that expose language models.

- [Foundry](#) is a platform that you use to build, test, and deploy AI solutions and models as a service (MaaS). This architecture uses Foundry to deploy an Azure OpenAI model.
 - [Foundry projects](#) establish connections to data sources, define agents, and invoke deployed models, including Azure OpenAI models. This architecture has only one Foundry project within the Foundry account.
 - [Agent Service](#) is a capability hosted in Foundry. You use this service to define and host agents to handle chat requests. It manages chat threads, orchestrates tool calls, enforces content safety, and integrates with identity, networking, and observability systems. In this architecture, Agent Service orchestrates the flow that fetches grounding data from AI Search and other connected knowledge sources and passes it with the prompt to the deployed model.

The agents defined in Agent Service are codeless and effectively nondeterministic. Your agent's system prompt, combined with `temperature` and `top_p` parameters, and constrained knowledge connections define how the agent behave for all requests.

- [Foundry Models](#) allow you to deploy flagship models, including OpenAI models, from the Azure AI catalog in a Microsoft-hosted environment. This approach is considered a MaaS deployment. This architecture deploys models by using the [Global Standard](#) configuration with a fixed quota.
- [AI Search](#) is a cloud search service that supports [full-text search](#), [semantic search](#), [vector search](#), and [hybrid search](#). This architecture includes AI Search because it's commonly used in orchestrations behind chat applications. You use AI Search to retrieve indexed

data relevant to user queries. AI Search serves as the knowledge store for the [Retrieval Augmented Generation](#) pattern. This pattern extracts a query from a prompt, queries AI Search, and uses the results as grounding data for a model.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

This basic architecture isn't intended for production. It favors simplicity and cost efficiency over functionality so that you can learn how to build end-to-end chat applications. The following sections outline deficiencies and recommendations. These omissions are deliberate to minimize setup time. Don't use this topology in production; each omission increases risk.

Reliability

Reliability helps ensure that your application can meet the commitments that you make to your customers. For more information, see [Design review checklist for Reliability](#).

The following list outlines critical reliability features that this architecture omits:

- This architecture uses the App Service Basic tier, which doesn't have [Azure availability zone](#) support. The instance becomes unavailable if there are problems with the instance, rack, or datacenter. As you move toward production, follow the [reliability guidance for App Service instances](#).
- This architecture doesn't enable autoscaling for the client UI. To avoid capacity issues, overprovision compute during learning. Implement autoscale before production.
- This architecture deploys Agent Service as a fully Microsoft-hosted solution. Microsoft hosts dependent services (Cosmos DB, Storage, AI Search) on your behalf. Your subscription doesn't show these resources. You don't control their reliability characteristics. For guidance on bringing your own dependencies, see the [baseline architecture](#).

ⓘ Note

The AI Search instance in the components section and diagram is different from the instance that's a dependency of Agent Service. The instance in the components

section stores your grounding data. The dependency does real-time chunking of files that are uploaded within a chat session or as part of an agent's definition.

- For learning, use the Global Standard model deployment type. Before production, estimate throughput and data residency needs. If you require reserved throughput, choose a [Data Zone Provisioned](#) or Global Provisioned deployment type. Use Data Zone Provisioned for explicit residency requirements.
- This architecture uses the AI Search Basic tier, which doesn't support [Azure availability zones](#). For zone redundancy, use the Standard tier or higher in a zone-enabled region and deploy three or more replicas.

For more information, see [Baseline Foundry chat reference architecture](#).

Security

Security provides assurances against deliberate attacks and the misuse of your valuable data and systems. For more information, see [Design review checklist for Security](#).

This section describes key recommendations that this architecture implements. These recommendations include content filtering and abuse monitoring, identity and access management, and role-based access control. This architecture isn't designed for production deployments, so this section also includes network security considerations. Network security is a key security feature that this architecture doesn't implement.

Content filtering and abuse monitoring

Foundry includes a [content filtering system](#) that uses a combination of classification models. This filtering detects and blocks specific categories of potentially harmful content in input prompts and output completions. This potentially harmful content includes hate, sexual content, self-harm, violence, profanity, and jailbreak (content designed to bypass language model restrictions) categories. You can configure the filtering strictness for each category by using low, medium, or high options. This reference architecture uses the `DefaultV2` content filter when deploying models. You should adjust the settings according to your requirements.

Identity and access management

The following guidance expands on the [identity and access management guidance](#) in the App Service baseline architecture. The chat UI uses its managed identity to authenticate the chat UI API code to Agent Service by using the Azure AI Persistent Agents SDK.

The Foundry project also has a managed identity. This identity authenticates to services such as AI Search through connection definitions. The project makes those connections available to Agent Service.

A Foundry account can contain multiple Foundry projects. Each project should use its own managed identity. If different workload components require isolated access to connected data sources, create separate Foundry projects within the same account and avoid sharing connections across them. If your workload doesn't require isolation, use a single project.

Role-based access roles

You're responsible for creating the required role assignments for the managed identities. The following table summarizes the role assignment that you must add to App Service, the Foundry project, and individuals who use the portal:

 [Expand table](#)

| Resource | Role | Scope |
|-----------------------------------|--------------------------|-----------------|
| App Service | Azure AI User | Foundry account |
| Foundry project | Search Index Data Reader | AI Search |
| Portal user (for each individual) | Azure AI Developer | Foundry account |

Network security

To simplify the learning experience for building an end-to-end chat solution, this architecture doesn't implement network security. It uses identity as its perimeter and uses public cloud constructs. Services such as AI Search, Foundry, and App Service are reachable from the internet. This setup increases the attack surface of the architecture.

This architecture also doesn't restrict egress traffic. For example, an agent can be configured to connect to any public endpoint based on the endpoint's OpenAPI specification. So data exfiltration of private grounding data can't be prevented through network controls.

For more information about network security as an extra perimeter in your architecture, see [networking in the baseline architecture](#).

If you want some network security during your evaluation of this solution, you should use the [network security perimeter support](#) on your Foundry project. This approach provides ingress and egress control before you implement virtual network resources in your architecture. When the Agent Service is configured for standard, private deployment, the network security perimeter is replaced with Private Link connections.

Microsoft Defender for Cloud

For this basic architecture, you don't need to enable Microsoft Defender cloud workload protection plans for any services. When you move to production, follow the [security guidance in the baseline architecture](#) for Microsoft Defender, which uses multiple plans to cover your workload.

Governance through policy

This architecture doesn't implement governance through Azure Policy. As you move toward production, follow the [governance recommendations in the baseline architecture](#). Those recommendations add Azure Policy across your workload's components.

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

This basic architecture doesn't represent the costs for a production-ready solution. It also doesn't include controls to guard against cost overruns. The following considerations outline crucial features that this architecture doesn't include. These features affect cost.

- This architecture assumes limited model calls. Use the Global Standard deployment type (pay-as-you-go) instead of provisioned throughput. As you move toward production, follow the [cost optimization guidance](#) in the baseline architecture.
- Agent Service incurs costs for files uploaded during chat interactions. Don't make file upload functionality available to application users if it's not part of the desired user experience. Extra knowledge connections, such as the [Grounding with Bing tool](#), have their own pricing structures.

Agent Service is a no-code solution. You can't deterministically control which tools or knowledge sources each request invokes. In cost modeling, assume maximum usage of each connection.

- This architecture uses the App Service Basic pricing tier on a single instance. It doesn't provide protection from an availability zone outage. The [baseline App Service architecture](#) recommends Premium plans with three or more worker instances for high availability.
- This architecture uses the AI Search Basic pricing tier with no added replicas. This topology can't withstand a zone failure. The [baseline end-to-end chat architecture](#) recommends the Standard tier or higher and three or more replicas.
- This architecture doesn't include cost governance or containment controls. Set Azure budgets and alerts early to guard against unexpected token or tool usage.

For budgeting, modify the [pricing calculator estimate](#) of this architecture to fit your scenario.

Operational Excellence

Operational Excellence covers the operations processes that deploy an application and keep it running in production. For more information, see [Design review checklist for Operational Excellence](#).

Monitoring

This architecture configures diagnostics for all services. App Service captures `AppServiceHTTPLogs`, `AppServiceConsoleLogs`, `AppServiceAppLogs`, and `AppServicePlatformLogs`. Foundry captures `RequestResponse`. During the POC phase, inventory available logs and metrics. Before production, remove sources that don't add value.

To use the monitoring capabilities in Foundry, [connect an Application Insights resource to your Foundry project](#).

This integration enables monitoring of:

- Real-time monitoring of token usage, including prompt, completion, and total tokens
- Detailed request-response telemetry, including latency, exceptions, and response quality

You can also [trace agents by using OpenTelemetry](#) for distributed diagnostics.

Model operations

This architecture is optimized for learning and isn't intended for production. Plan for model lifecycle management and [model deprecation and retirement](#) before promoting workloads.

Development

For the basic architecture, you can create agents by using the browser-based experience in the Foundry portal. When you move toward production, follow the [development and source control guidance](#) in the baseline architecture. When you no longer need an agent, be sure to delete it. If the agent that you delete is the last one that uses a connection, also remove the connection.

Evaluation

Evaluate your generative application in Foundry. Learn how to [use evaluators](#). This helps ensure model, prompt, and data quality meet design requirements.

Performance Efficiency

Performance Efficiency refers to your workload's ability to scale to meet user demands efficiently. For more information, see [Design review checklist for Performance Efficiency](#).

This architecture isn't designed for production deployments, so it omits critical performance efficiency features:

- Use POC results to choose the right App Service product. Meet demand through horizontal scaling (adjust instance count). Avoid designs that require changing the product tier to handle routine demand.
- This architecture uses pay-as-you-go components. Best-effort resource allocation can introduce noisy neighbor effects. Decide whether you need [provisioned throughput](#) to reserve capacity and achieve predictable performance.

Other design recommendations

Architects should design AI and machine learning workloads, such as this one, with the Well-Architected [AI workloads on Azure](#) guidance in mind. Combine insights from your POC by using this architecture with broader AI and machine learning best practices when you move beyond POC.

Deploy this scenario

[Deploy a reference implementation](#) ↗ that applies these recommendations and considerations.

Next step

Related resources

- [A Well-Architected Framework perspective on AI workloads on Azure](#)
- [Deploy AI models in the Foundry portal](#)
- [Explore Models](#)
- [What is Agent Service?](#)

Baseline Microsoft Foundry chat reference architecture

Azure AI services

Azure App Service

Azure Monitor

Azure AI Foundry

Azure AI Foundry Agent Service

Enterprise chat applications can empower employees through conversational interactions with AI agents. This capability is increasingly powerful thanks to ongoing advancements in language models, such as OpenAI's GPT models and orchestration SDKs like the Microsoft Agent Framework. These chat applications typically consist of the following components:

- A chat user interface (UI) that's integrated into a larger enterprise application. It provides users with a conversational experience alongside other business functions.
- Data repositories that contain domain-specific information that's pertinent to user queries.
- Language models that reason over the domain-specific data to produce relevant responses.
- An orchestrator or agent that oversees the interactions between data sources, language models, and the end user.

This article provides a baseline architecture to help you build and deploy enterprise chat applications by using [Microsoft Foundry](#) and [Azure OpenAI in Foundry Models](#). This architecture uses a single agent hosted in Foundry Agent Service. The agent receives user messages and then queries data stores to retrieve grounding information for the language model.

The chat UI follows the [baseline Azure App service web application](#) guidance about how to deploy a secure, zone-redundant, and highly available web application on App Service. In that architecture, App Service communicates with the Azure platform as a service (PaaS) solution through virtual network integration over private endpoints. In the chat UI architecture, App Service communicates with the agent over a private endpoint. Public access to the Foundry portal and agents is disabled.

Important

This article doesn't describe components or architecture decisions from the [baseline App Service web application architecture](#). Read that article for architectural guidance about how to host the web application that contains your chat UI.

This architecture uses the [Agent Service standard agent setup](#) to enable enterprise-grade security, compliance, and control. In this configuration, you bring your own network for

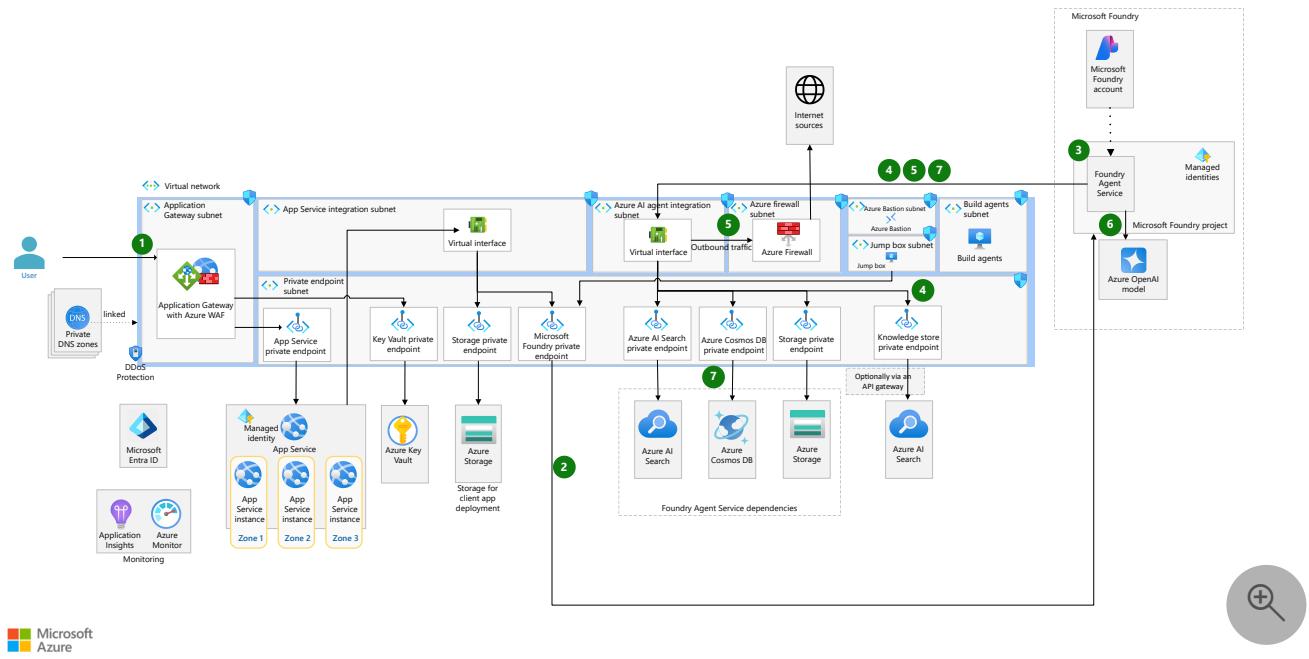
network isolation and your own Azure resources to store chat and agent state. All communication between application components and Azure services occurs over private endpoints, which ensures that data traffic remains within your workload's virtual network. Outbound traffic from the agents strictly routes through Azure Firewall, which enforces egress rules.

Tip



The [Agent Service reference implementation](#) showcases a baseline end-to-end chat implementation on Azure. It serves as a foundation to develop custom solutions as you move toward production.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

1. An application user interacts with a chat UI. The requests are routed through Azure Application Gateway. Azure Web Application Firewall inspects these requests before it forwards them to the back-end App Service.
2. When the web application receives a user query or instruction, it invokes the purpose-built agent. The web application communicates with the agent via the Azure AI Agent.

SDK. The web application calls the agent over a private endpoint and authenticates to Foundry by using its managed identity.

3. The agent processes the user's request based on the instructions in its system prompt. To fulfill the user's intent, the agent has a configured language model, connected tools, and connected knowledge stores.
4. The agent connects to the knowledge store (Azure AI Search) in the private network via a private endpoint.
5. Requests to most external knowledge stores or tools, such as Wikipedia, traverse Azure Firewall for inspection and egress policy enforcement.
6. The agent connects to its configured language model and passes relevant context.
7. Before the agent returns the response to the UI, it persists the request, the generated response, and a list of consulted knowledge stores into a dedicated *memory* database. This database maintains the complete conversation history, which enables context-aware interactions and allows users to resume conversations with the agent without losing prior context.

The Foundry APIs support the development of user experiences that manage multiple concurrent, context-isolated conversations.

Components

This architecture builds on the [basic Foundry chat reference architecture](#). This architecture introduces more Azure services to address enterprise requirements for reliability, security, and operational control. Each of the following components plays a specific role in a production enterprise chat solution:

- [Agent Service](#) is a cloud-native runtime environment that enables intelligent agents to operate securely and autonomously. In this architecture, Agent Service provides the orchestration layer for chat interactions. It hosts and manages agents that do the following tasks:
 - Process user requests
 - Coordinate calls to tools and other agents
 - Enforce content safety
 - Integrate with enterprise identity, networking, and observability

The [standard agent setup](#) ensures network isolation and provides control over data storage. You supply dedicated Azure resources for agent state, chat history, and file

storage, which Agent Service manages exclusively. Other application components in the workload shouldn't use these required resources.

- [Azure Cosmos DB for NoSQL](#) is a globally distributed, document database service. In this architecture, it hosts the workload's memory database, called `enterprise_memory`, within your subscription. This database stores the agent's operational state, including chat threads and agent definitions. This design ensures that chat history and agent configuration are isolated, auditable, and managed according to data governance requirements. Agent Service manages the database, its collections, and its data.
- [Azure Storage](#) is a cloud storage service for unstructured data. In this architecture, it provides dedicated storage for files uploaded during chat sessions. Hosting this account in your subscription provides granular access control, auditing capabilities, and compliance with data residency or retention policies. Agent Service manages the containers and data life cycle within those containers.
- [AI Search](#) is a search-as-a-service cloud solution that provides rich search capabilities. In this architecture, it stores a searchable, chunked index of uploaded files from conversations with the agent. AI Search also stores chunked, static files that are added as knowledge sources when the agent is created to be used across all agent invocations. Agent Service manages the index, schema, and data, and uses its own chunking strategy and query logic.
- [Application Gateway](#) is a web traffic load balancer and application delivery controller. In this architecture, it acts as a secure, scalable entry point for all HTTP and HTTPS traffic to the chat UI. It also provides Transport Layer Security (TLS) termination and path-based routing. Application Gateway distributes requests across availability zones, which supports high availability and performance for the web application tier. Its back end is the App Service instance that hosts the application code.
 - [Azure Web Application Firewall](#) is a cloud-native service that protects web applications from common web exploits. In this architecture, it integrates with Application Gateway to protect the chat UI from common web vulnerabilities and attacks. It inspects and filters HTTP requests, which provides a security layer for public-facing applications.
 - [Azure Key Vault](#) is a cloud service for securely storing and accessing secrets, keys, and certificates. In this architecture, it securely stores and manages the TLS certificates that Application Gateway requires. Centralized certificate management in Key Vault supports automated rotation, auditing, and compliance with organizational security standards. This architecture doesn't require stored keys or other secrets.
- [Azure Virtual Network](#) is the fundamental building block for private networks in Azure. In this architecture, it provides network isolation for all components to help you meet

security and compliance requirements. When you place resources in a private virtual network, you control east-west and north-south traffic, enforce segmentation, keep traffic private, and enable inspection of ingress and egress flows.

- [Azure Private Link](#) provides private connectivity from a virtual network to Azure PaaS services. In this architecture, it connects all PaaS services, such as Azure Cosmos DB, Storage, AI Search, and Agent Service, to the virtual network via private endpoints. Private Link helps you ensure that all data traffic remains on the Azure backbone, which eliminates exposure to the public internet and reduces the attack surface.
- [Azure Firewall](#) is a managed, cloud-based network security service. In this architecture, it inspects and controls all outbound (egress) traffic from the virtual network. It also enforces fully qualified domain name (FQDN)-based rules, which ensures that only approved destinations are reachable. This configuration helps prevent data exfiltration and meet requirements for network security.
- [Azure DNS](#) provides private Domain Name System (DNS) zones linked to the virtual network. In this architecture, it enables name resolution for private endpoints, which ensures that all service-to-service communication uses private IP addresses and remains within the network boundary.
- [Storage](#) supports secure, automated deployment workflows and separates application artifacts from compute resources. In this architecture, it hosts the web application code as a ZIP file for deployment to App Service.

Alternatives

This architecture includes multiple components that you can substitute with other Azure services or approaches, depending on your workload's functional and nonfunctional requirements. Consider the following alternatives and trade-offs.

Chat orchestration

Current approach: This architecture uses [Agent Service](#) to orchestrate chat flows, including fetching grounding data from knowledge stores and invoking Azure OpenAI models. Agent Service provides codeless, nondeterministic orchestration. It handles chat requests, thread management, tool invocation, content safety, and integration with identity, networking, and observability. It provides a native memory database solution.

Alternative approach: You can self-host the orchestration layer by using frameworks such as [Microsoft Agent Framework](#), [Semantic Kernel](#), or [LangChain](#). Use these frameworks to implement deterministic, code-driven chat flows and custom orchestration logic.

Consider this alternative if your workload requires the following capabilities:

- The use of a model other than those [models supported by the Agent Service](#)
- Fine-grained, deterministic control over the orchestration sequence, tool invocation, or prompt engineering
- Integration with custom business logic or external systems that Agent Service doesn't natively support
- Advanced client request routing for experimentation or safe deployment practices
- A custom memory database solution that differs from the native Agent Service solution

Self-hosted orchestration increases operational complexity and requires you to manage compute, scaling, and security.

Application tier components

Current approach: The chat UI front-end website is hosted on the Web Apps feature of App Service, which provides a managed, scalable platform for web applications. Web Apps integrates natively with Azure networking and security features.

Alternative approach: You can use other Azure-managed compute platforms, such as Azure Container Apps or Azure Kubernetes Service (AKS), to host the application tier.

Consider this alternative if your workload meets any of the following conditions:

- Another compute platform better supports certain use cases, and colocating services on that platform can improve cost efficiency and simplify operations
- Your application requires more sophisticated scaling, orchestration, or custom networking

App Service remains the preferred option for its simplicity in hosting web applications and their APIs.

Grounding data (knowledge) store

Current approach: This architecture uses AI Search as the primary data store for grounding knowledge. It uses AI Search vector search and semantic ranking capabilities.

Alternative approach: You might use other data platforms for grounding knowledge, such as Azure Cosmos DB, Azure SQL Database, or other online transaction processing (OLTP) data stores. Your data platform depends on your existing data estate, data freshness requirements, and query requirements.

Consider this alternative if your workload meets any of the following conditions:

- You already manage your grounding knowledge in an existing transactional or operational database
- You require multi-model or SDK support not available in AI Search
- You need to integrate with specialized data sources or legacy systems

Vector search is common for retrieval-augmented generation but not always required. For more information, see [Choose an Azure service for vector search](#). Before you choose a data store, evaluate the data access patterns, latency, and scalability needs of your workload.

Predefined agent or dynamically created agent

Current approach: The reference implementation uses a statically defined agent that's deployed as a microservice within Foundry. The agent's logic and data sources are configured at deployment and remain unchanged until the next application release. This approach works well when agent behavior and data sources are stable and controlled through DevOps processes.

Alternative approach: You can dynamically create or modify agents at runtime by using the Azure AI Agent SDK. This approach allows the application to instantiate agents on demand, adjust system prompts, or reconfigure connections based on user context or business logic.

Consider dynamic agents if your workload requires the following capabilities:

- Personalized agent behavior or data sources for each user or session
- Frequent or programmatic changes to agent configuration
- Ephemeral, context-specific agent support for advanced user experiences

Dynamic agent management increases flexibility but also introduces the burden of life cycle management. Ensure that you have appropriate controls for agent creation, modification, and cleanup.

Choose the agent approach that aligns with your workload's user experience requirements.

Single-agent or multi-agent orchestration

Current approach: This reference architecture uses a single agent that has access to all necessary knowledge sources and tools to handle most user interactions effectively.

Alternative approach: You can orchestrate multiple specialized agents, where each agent focuses on specific domains, uses different models, or accesses distinct knowledge stores and tools.

Consider a multi-agent approach when your workload exhibits the following characteristics:

- Requests span multiple expertise areas, such as financial analysis, legal review, and technical implementation. Specialized agents provide deeper, more accurate responses within their respective domains.
- Information requires different permission levels. An HR agent might access employee data, while a customer service agent accesses only product information. Multi-agent architectures enable granular security boundaries at the agent level.
- Different query interactions benefit from different models. A lightweight model handles simple questions, while a more powerful model processes complex reasoning tasks. This approach optimizes both cost and latency.
- The chat experience serves as a front end to business processes that involve sequential or parallel steps that require different specialists.

Multi-agent approaches introduce coordination complexity and increased latency because of communication between agents. Use a single agent when your use case is well-defined, doesn't require strict access isolation, and can be handled effectively by one model with a reasonable set of tools.

For guidance about how to implement multiple coordinated agents, see [AI agent orchestration patterns](#). This article covers sequential, concurrent, group chat, handoff, and magentic orchestration approaches. You can implement some patterns within Agent Service. Other patterns require self-hosted orchestration by using an SDK such as Semantic Kernel.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

Apply this architecture and the [AI workloads on Azure design guidance](#) during the design phase of your workload.

Reliability

Reliability helps ensure that your application can meet the commitments that you make to your customers. For more information, see [Design review checklist for Reliability](#).

The baseline App Service web application architecture focuses on zonal redundancy for key regional services. Availability zones are physically separate locations within a region that provide redundancy when you deploy two or more instances across them. If one zone experiences downtime, others in the region might remain unaffected. The architecture also distributes instances and configurations of Azure services across availability zones. For more information, see [Baseline highly available zone-redundant web application](#).

This section addresses reliability for components not covered in the App Service baseline architecture, specifically Foundry and AI Search.

Zone redundancy in your orchestration layer

Enterprise deployments usually require zonal redundancy to minimize the risk of service disruption from zone-level failures. In Azure, zonal redundancy means that you use resources that support [availability zones](#) and deploy at least three instances, or enable platform-level redundancy where direct instance control is unavailable.

In this architecture, Foundry hosts the Agent Service capability. The agent's reliability depends on the availability of the Agent Service dependencies, which are Azure Cosmos DB, Storage, and AI Search. Agent Service manages the data within these services, but you configure their reliability in your subscription.

To achieve zonal redundancy for the orchestration layer, follow these recommendations:

- Enable [zone redundancy in your Azure Cosmos DB for NoSQL account](#). This configuration ensures synchronous data replication across multiple zones, which reduces the risk of data loss or downtime from a single-zone failure.

Also consider [global distribution](#) to mitigate regional outages within Azure Cosmos DB.

- Use [zone-redundant storage \(ZRS\)](#) for your Storage account. For higher resilience, use [geo-zone-redundant storage \(GZRS\)](#), which combines zonal and regional redundancy.
- Configure your [AI Search instance](#) with at least three replicas. This configuration ensures that the service distributes replicas across unique zones in your region.

If your agent integrates with other workload-specific dependencies, such as custom tool connections or external knowledge stores, ensure that those dependencies meet your availability and redundancy requirements. Any single-zone or nonredundant dependency can undermine the overall reliability of the orchestration layer.

The Foundry portal, its data plane APIs, and the Agent Service capability don't provide direct controls for zone redundancy.

Reliability in Foundry model hosting

Foundry provides models as a service (MaaS) hosting with several deployment options. These options primarily support quota and throughput management, rather than traditional high availability within a region. Standard model deployments operate in a single region and don't support availability zones. To achieve multi-datacenter availability, you must use either a global or data zone model deployment.

For enterprise chat scenarios, deploy both a [data zone provisioned](#) and [data zone standard](#) model. Configure [spillover](#) to handle excess traffic or failures. If your workload doesn't require low latency or strict geographic data residency and processing, use global deployments for maximum resilience.

Foundry doesn't support advanced load balancing or failover mechanisms, such as round-robin routing or [circuit breaking](#), for model deployments. If you require granular redundancy and failover control within a region, host your model access logic outside the managed service. For example, you can build a custom gateway by using Azure API Management. This approach allows you to implement custom routing, health checks, and failover strategies. But it also increases operational complexity and shifts responsibility for the reliability of that component to your team.

You can also expose gateway-fronted models as custom API-based tools or knowledge stores for your agent. For more information, see [Use a gateway in front of multiple Azure OpenAI deployments or instances](#).

Reliability in AI Search for enterprise knowledge

Deploy AI Search by using the Standard pricing tier or higher in a [region that supports availability zones](#). Configure at least three replicas to ensure that the service distributes instances across separate availability zones. This configuration provides resilience to zone-level failures and supports high availability for search operations.

To determine the optimal number of replicas and partitions for your workload, use the following methods:

- [Monitor AI Search](#) by using built-in metrics and logs. Track query latency, throttling, and resource usage.
- Use monitoring metrics and logs and performance analysis to determine the appropriate number of replicas. This approach helps you avoid throttling from high query volume,

insufficient partitions, or index limitations.

- Ensure indexing reliability by avoiding disruptions from periodic indexing or indexing errors. Consider indexing into an offline index and swapping from your live index to your rebuilt index after you validate data integrity.

Reliability in Azure Firewall

Azure Firewall is a critical egress control point in this architecture but represents a single point of failure for all outbound traffic. To mitigate this risk, deploy Azure Firewall [across all availability zones](#) in your region. This configuration helps maintain outbound connectivity if a zone becomes unavailable.

If your workload requires a high volume of concurrent outbound connections, configure Azure Firewall with multiple public IP addresses. This approach distributes Source Network Address Translation (SNAT) connections across [multiple IP address prefixes](#), which reduces the risk of SNAT port exhaustion. [SNAT exhaustion](#) can cause intermittent or total loss of outbound connectivity for agents and other workload components, which can result in feature downtime or degraded performance.

Monitor [SNAT port usage and firewall health](#). If you observe connection failures or throughput problems, review firewall metrics and logs to identify and address SNAT exhaustion or other bottlenecks.

Isolate Agent Service dependencies from similar workload components

To maximize reliability and minimize the blast radius of failures, strictly isolate the Agent Service dependencies from other workload components that use the same Azure services. Specifically, don't share AI Search, Azure Cosmos DB, or Storage resources between the agent service and other application components. Instead, provision dedicated instances for the agent service's required dependencies.

This separation provides two key benefits:

- It contains failures or performance degradation to a single workload segment, which prevents cascading effects across unrelated application features.
- It enables you to apply targeted operational processes, such as backup, restore, and failover. These processes are tuned to the specific availability and recovery requirements of the workload flow that uses those resources.

For example, if your chat UI application needs to store transactional state in Azure Cosmos DB, provision a separate Azure Cosmos DB account and database for that purpose, rather than

reusing the account or database that Agent Service manages. Even if cost or operational simplicity motivates resource sharing, the risk of a reliability event affecting unrelated workload features outweighs the potential savings in most enterprise scenarios.

Important

If you colocate workload-specific data with the agent's dependencies for cost or operational reasons, never interact directly with the system-managed data, such as collections, containers, or indexes, that Agent Service creates. These internal implementation details are undocumented and subject to change without notice. Direct access can break the agent service or result in data loss. Always use the Agent Service data plane APIs for data manipulation, such as fulfilling right to be forgotten (RTBF) requests. Treat the underlying data as opaque and monitor-only.

Multi-region design

This architecture uses availability zones for high availability within a single Azure region. It's not a multi-region solution. It lacks the following critical elements required for regional resiliency and disaster recovery (DR):

- Global ingress and traffic routing
- DNS management for failover
- Data replication or isolation strategies across regions
- An active-active, active-passive, or active-cold designation
- Regional failover and fallback processes to meet recovery time objectives (RTOs) and recovery point objectives (RPOs)
- Consideration of region availability for developer experiences, including the Foundry portal and data plane APIs

If your workload requires business continuity if a regional outage occurs, you must design and implement extra components and operational processes beyond this architecture. Specifically, you need to address load balancing and failover at each architectural layer, including the following areas:

- Grounding data (knowledge stores)
- Model hosting and inference endpoints
- The orchestration or agent layer
- User-facing UI traffic and DNS entry points

You must also ensure that observability, monitoring, and content safety controls remain operational and consistent across regions.

This baseline architecture lacks multi-region capabilities, so regional outages are likely to result in loss of service within your workload.

Disaster recovery

Chat architectures contain stateful components, so DR planning is essential. These workloads typically require a memory store for active or paused chat sessions. They also require storage for supplemental data, such as documents or images, added to chat threads. The agent orchestration layer might also maintain state that's specific to conversation flows. In this architecture, Agent Service uses Azure Cosmos DB, Storage, and AI Search to persist operational and transactional state. The life cycle and coupling of this state across components determines your DR strategy and recovery operations.

For Agent Service, ensure that you can recover all dependencies to a consistent point in time. This approach helps maintain transactional integrity across the three external dependencies.

The following recommendations are key excerpts from the [Agent Service disaster recovery](#) guide:

- **Azure Cosmos DB:** Enable [continuous backup](#) for the `enterprise_memory` database. This setup provides point-in-time restore (PITR) with a seven-day RPO, which includes agent definitions and chat threads. Test your restore process regularly to confirm that it meets your RTO and that the restored data remains available to the agent service. Always restore to the same account and database.
- **AI Search:** AI Search lacks built-in restore capabilities and doesn't support direct index manipulation. If data loss or corruption occurs, you must contact Microsoft support for assistance with index restoration. This limitation can significantly affect your RTO. If your chat UI doesn't support file uploads and you don't have agents that use static files as knowledge, you might not need a DR plan for AI Search.

Maintain a separate, regularly updated source of truth for your enterprise grounding knowledge. This practice ensures that you can rebuild indexes when necessary.

- **Storage:** If you have a geo-redundant storage account, use [customer-managed failover](#) for blob containers that Agent Service uses. This setup allows you to initiate failover during a regional outage. If you use only zone-redundant storage, contact Microsoft support to restore data. This process might extend your RTO. As with AI Search, if your chat UI doesn't support file uploads and you don't have agents that use static files as knowledge, you might not need a DR plan for blob containers.
- **Transactional consistency:** If the state store in your workload references Azure AI agent identifiers, such as thread IDs or agent IDs, coordinate recovery across all relevant data

stores. Restoring only a subset of dependencies can result in orphaned or inconsistent data. Design your DR processes to maintain referential integrity between your workload and the agent service's state.

- **Agent definitions and configuration:** Define agents *as code*. Store agent definitions, connections, system prompts, and parameters in source control. This practice enables you to redeploy agents from a known good configuration if you lose the orchestration layer. Avoid making untracked changes to agent configuration through the Foundry portal or data plane APIs. This approach ensures that your deployed agents remain reproducible.
- **Foundry projects:** Use a user-assigned managed identity for a project's identity. If you need to recover a project when it's accidentally deleted, having a user managed identity on that project will allow you to reuse your existing role assignments when you re-create your project and its capability host, minimizing change coordination with all project dependencies.

As an added preventative measure for the Agent Service dependencies, we recommend you add a *delete* resource lock to each service. This will help protect against catastrophic loss of state in AI Search, Cosmos DB, and Storage.

Before you move to production, build a recovery runbook that addresses failures in both application-owned state and agent-owned state.

Security

Security provides assurances against deliberate attacks and the misuse of your valuable data and systems. For more information, see [Design review checklist for Security](#).

This architecture extends the security foundation established in the [basic Foundry chat reference architecture](#). The primary difference is the addition of a network security perimeter alongside the identity perimeter from the basic architecture. From a network perspective, Application Gateway is the only internet-exposed resource. It makes the chat UI application available to users. From an identity perspective, the chat UI should authenticate and authorize requests. Use managed identities when possible to authenticate applications to Azure services.

Identity and access management

This architecture primarily uses system-assigned managed identities for service-to-service authentication. You might also use user-assigned managed identities. In either case, apply the following principles:

- Isolate identities by resource and function. Provision distinct managed identities for the following components:

- The Foundry account
- Each Foundry project
- The web application
- Any custom orchestrator or integration code
- Assign an identity to an Azure resource only if that resource must authenticate as a client to another Azure service.
- Use fit-for-purpose identity types. Where possible, use [workload identities](#) for applications and automation, and use [agent identities](#) for AI agents.

Connections

Connections define how a Foundry account or an individual project authenticates to and uses an [external dependency](#). Create connections at the project level when possible. Remove unused connections. Prefer Microsoft Entra ID-based authentication for all connections.

If Microsoft Entra ID isn't supported for a connection, you must supply a secret (for example, an API key). Store these secrets in a dedicated, self-hosted Azure Key Vault. Configure an [Azure Key Vault connection](#) for the Foundry account so the service can read and write the secrets it manages.

Use this Key Vault only for Foundry. Don't share it with other workload components. All non-Entra ID connections used across all projects in the account store their secrets in this single vault. Additional components in your workload do not need access to these secrets to consume Foundry capabilities and shouldn't be granted permission to read or write to this vault unless a clear operational requirement exists or tradeoff is desired.

In this architecture, there are two API key based connections: Application Insights for Foundry metrics and Grounding with Bing Search.

If you use customer-managed keys (CMK) for encryption, you can host both the CMK keys and the connection secrets in the same dedicated vault, subject to your security governance policies concerning this.

Foundry portal employee access

When you onboard employees to Foundry projects, assign the minimum permissions required for their role. Use Microsoft Entra ID groups and Azure role-based access control (Azure RBAC) to enforce separation of duties. For example, distinguish agent developers from data scientists who handle fine-tuning tasks. However, be aware of the limitations and risks.

The Foundry portal runs many actions by using the service's identity rather than the employee's identity. As a result, employees that have limited Azure RBAC roles might have visibility into sensitive data, such as chat threads, agent definitions, and configuration. This Foundry portal design can inadvertently bypass your desired access constraints and expose more information than intended.

To mitigate the risk of unauthorized access, restrict portal usage in production environments to employees that have a clear operational need. For most employees, disable or block access to the Foundry portal in production. Instead, use automated deployment pipelines and infrastructure as code (IaC) to manage agent and project configuration.

Treat creating new projects in a Foundry account as a privileged action. Projects created through the portal don't automatically inherit your established network security controls, such as private endpoints or network security groups (NSGs). And new agents in those projects bypass your intended security perimeter. Enforce project creation exclusively through your controlled, auditable IaC processes.

Foundry project role assignments and connections

To use Agent Service in Standard mode, the project must have administrative permissions on the Agent Service dependencies. Specifically, the project's managed identity must have elevated role assignments on the Storage account, AI Search, and the Azure Cosmos DB account. These permissions provide nearly full access to these resources, including the ability to read, write, modify, or delete data. To uphold least privilege access, isolate your workload resources from the Agent Service dependencies.

All agents within a single Foundry project share the same managed identity. If your workload uses multiple agents that require access to different sets of resources, the principle of least privilege requires you to create a separate Foundry project for each distinct agent access pattern. This separation allows you to assign only the minimum required permissions to each project's managed identity, which reduces the risk of excessive or unintended access.

When you establish [connections](#) to external resources from within Foundry, use Microsoft Entra ID-based authentication if available. This approach eliminates the need to maintain preshared secrets. Scope each connection so that only the owning project can use it. If multiple projects require access to the same resource, create a separate connection in each project rather than sharing a single connection across projects. This practice enforces strict access boundaries and prevents future projects from inheriting access that they don't require.

Avoid creating connections at the Foundry account level. These connections apply to all current and future projects in the account. They can inadvertently grant broad access to resources,

violate least privilege principles, and increase the risk of unauthorized data exposure. Prefer project-level connections only.

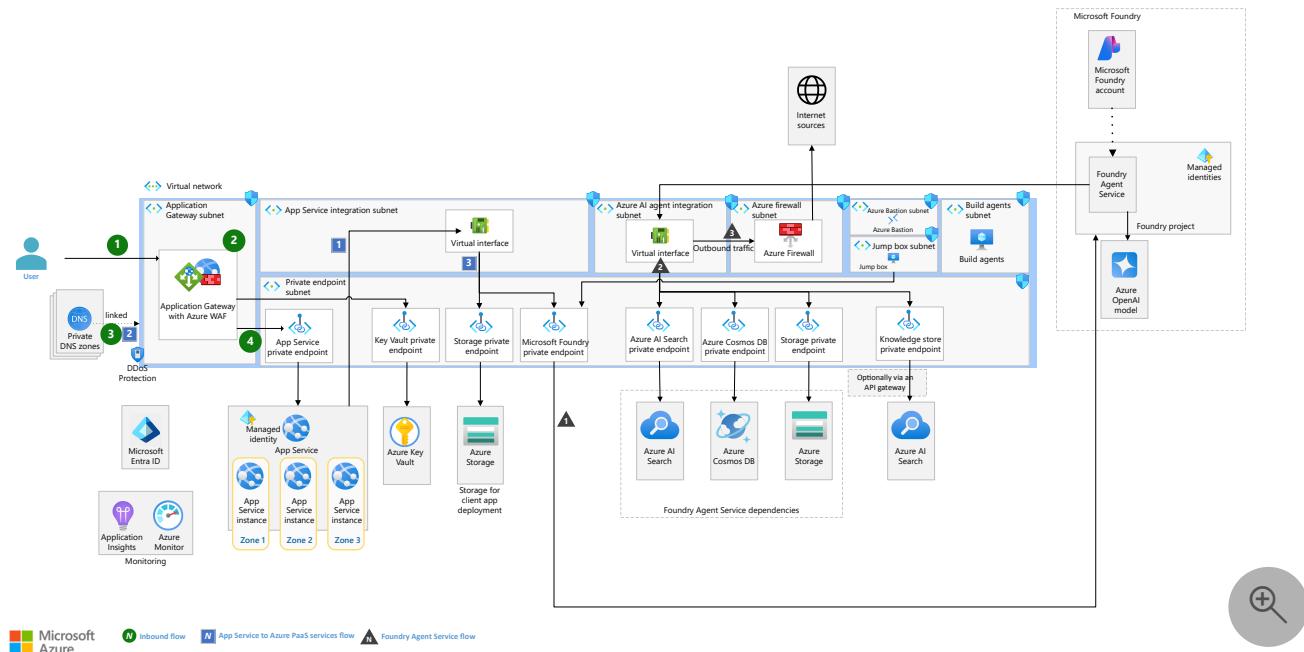
Networking

In addition to identity-based access, this architecture requires network confidentiality.

The network design includes the following safeguards:

- A single, secure entry point for all chat UI traffic, which minimizes the attack surface
- Filtered ingress and egress network traffic by using a combination of NSGs, a web application firewall, user-defined routes (UDRs), and Azure Firewall rules
- End-to-end encryption of data in transit by using TLS
- Network privacy by using Private Link for all Azure PaaS service connections
- Logical segmentation and isolation of network resources, with dedicated subnets for each major component grouping to support granular security policies

Network flows



The [baseline App Service web application architecture](#) outlines the inbound flow from the user to the chat UI and the flow from App Service to [Azure PaaS services](#). This section focuses on agent interactions.

When the chat UI communicates with the agent deployed in Foundry, the following network flows occur:

1. The App Service-hosted chat UI initiates HTTPS requests through a private endpoint to the Foundry data plane API endpoint.
2. When the agent accesses Azure PaaS services, such as service dependencies, custom knowledge stores, or custom tools, it sends HTTPS requests from the delegated subnet to the private endpoints of those services.
3. When the agent accesses resources outside the virtual network, including internet-based APIs or external services, it's forced to route those HTTPS requests from the delegated subnet through Azure Firewall.

Private endpoints serve as a critical security control in this architecture by supplementing identity-based security. Because this architecture uses private endpoints and UDRs in your virtual network, the [network security perimeter](#) capability of Foundry projects isn't supported.

Ingress to Foundry

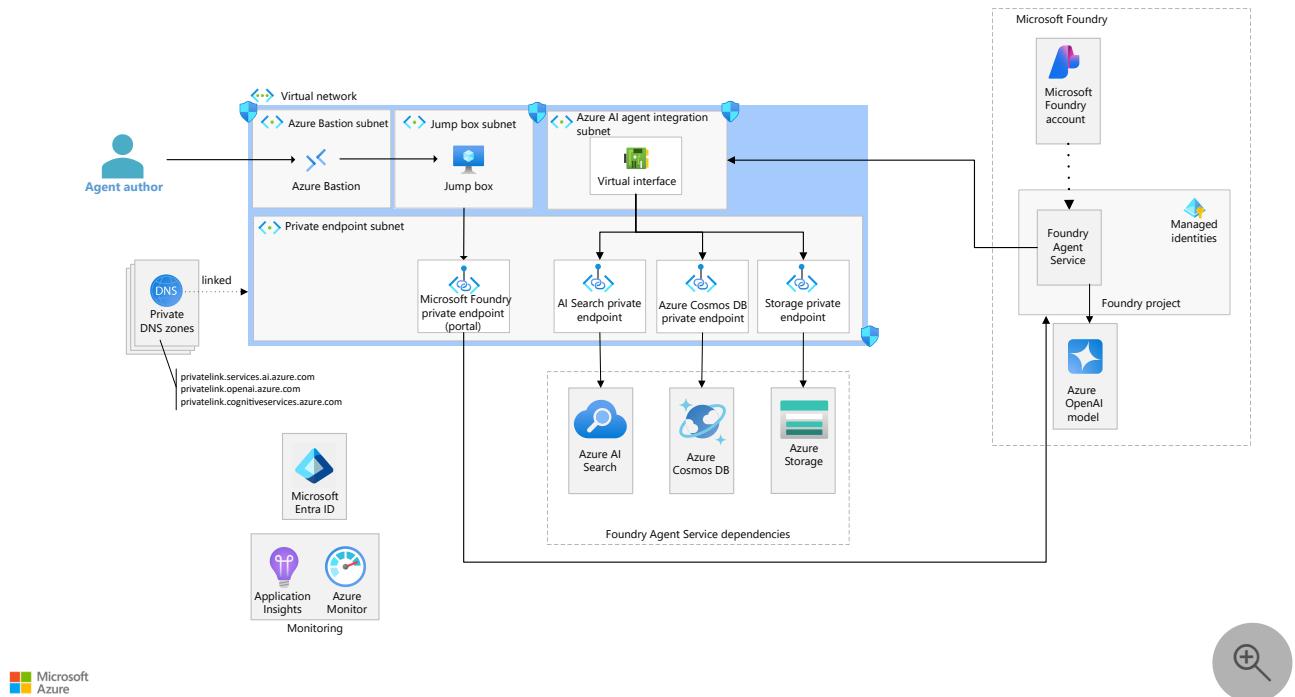
This architecture disables public access to the Foundry data plane by only allowing traffic through a [private link for Foundry](#). You can access much of the Foundry portal through the [portal website](#), but all project-level functionality requires network access. The portal relies on your Foundry account's data plane APIs, which are reachable only through private endpoints. As a result, developers and data scientists must access the portal through a jump box, a peered virtual network, or an ExpressRoute or site-to-site VPN connection.

All programmatic interactions with the agent data plane, such as from the web application or from external orchestration code when invoking model inferencing, must also use these private endpoints. Private endpoints are defined at the account level, not the project level. Therefore, all projects within the account share the same set of endpoints. You can't segment network access at the project level, and all projects share the same network exposure.

To support this configuration, set up DNS for the following Foundry FQDN API endpoints:

- `privatelink.services.ai.azure.com`
- `privatelink.openai.azure.com`
- `privatelink.cognitiveservices.azure.com`

The following diagram shows how an AI developer connects through Azure Bastion to a virtual machine (VM) jump box. From that jump box, the author can access the project in the Foundry portal through a private endpoint in the same network.



Control traffic from the Foundry agent subnet

This architecture routes all outbound (egress) network traffic from the Agent Service capability through a delegated subnet within your virtual network. This subnet serves as the sole egress point for both the agent's required three service dependencies and any external knowledge sources or tool connections that the agent uses. This design helps reduce data exfiltration attempts from within the orchestration logic.

By forcing this egress path, you gain full control over outbound traffic. You can apply granular NSG rules, custom routing, and DNS control to all agent traffic that leaves the service.

The agent service uses the virtual network's DNS configuration to resolve private endpoint records and required external FQDNs. This setup ensures that the agent's requests generate DNS logs, which support auditing and troubleshooting.

The NSG attached to the agent egress subnet blocks all inbound traffic because no legitimate ingress should occur. Outbound NSG rules allow access only to private endpoint subnets within the virtual network and to Transmission Control Protocol (TCP) port 443 for internet-bound traffic. The NSG denies all other traffic.

To further restrict internet traffic, this architecture applies a UDR to the subnet, which directs all HTTPS traffic through Azure Firewall. The firewall controls which FQDNs the agent can reach through HTTPS connections. For example, if the agent only needs to access <https://example.org/api>, configured Azure Firewall to allow traffic to `api.example.org` on port 443 from this subnet and ensure the NSG allows that traffic as well.

(!) Note

Not all knowledge tools connected to your agents egress through this subnet. For example, [Grounding with Bing](#) public APIs ideally would be configured in your Azure Firewall to allow traffic to `api.bing.microsoft.com` on port 443 from this subnet. However, that specific tool is invoked from within the agent service through a mechanism that doesn't use the egress subnet. Test all built-in knowledge and tool connections you consider for your workload to see if they align with your network egress control policies.

Virtual network segmentation and security

This architecture segments the virtual network by assigning each major component group to its own subnet. Each subnet has a dedicated NSG that limits the inbound and outbound traffic to only what the component requires.

The following table summarizes the NSG and firewall configuration for each subnet.

[\[+\] Expand table](#)

| Usage or subnet name | Inbound traffic (NSG) | Outbound traffic (NSG) | UDR to firewall | Firewall egress rules |
|---|---|--|-----------------|-----------------------|
| Private endpoints <code>snet-privateEndpoints</code> | Virtual network | No traffic allowed | Yes | No traffic allowed |
| Application Gateway <code>snet-appGateway</code> | Chat UI user source IP addresses, such as the public internet, and required sources for the service | Private endpoint subnet and required items for the service | No | - |
| App Service <code>snet-appServicePlan</code> | No traffic allowed | Private endpoints and Azure Monitor | Yes | To Azure Monitor |

| Usage or subnet name | Inbound traffic (NSG) | Outbound traffic (NSG) | UDR to firewall | Firewall egress rules |
|--|--|--|-----------------|---|
| Agent Service <code>snet-agentsEgress</code> | No traffic allowed | Private endpoints and the internet | Yes | Only public FQDNs that you allow your agents to use |
| Jump box VMs <code>snet-jumpBoxes</code> | Azure Bastion subnet | Private endpoints and the internet | Yes | As needed by the VM |
| Build agents <code>snet-buildAgents</code> | Azure Bastion subnet | Private endpoints and the internet | Yes | As needed by the VM |
| Azure Bastion <code>AzureBastionSubnet</code> | See NSG access and Azure Bastion | See NSG access and Azure Bastion | No | - |
| Azure Firewall <code>AzureFirewallSubnet</code> <code>AzureFirewallManagementSubnet</code> | No NSG | No NSG | No | - |

This design explicitly denies all other traffic, either through NSG rules or by default in Azure Firewall.

When you implement network segmentation and security in this architecture, follow these recommendations:

- Deploy an [Azure DDoS Protection](#) plan on the Application Gateway public IP address to mitigate volumetric attacks.
- Attach an [NSG](#) to every subnet that supports it. Apply the strictest rules possible without disrupting required functionality.

- Apply [forced tunneling](#) to all supported subnets so that your egress firewall can inspect all outbound traffic. Use forced tunneling even on subnets where you don't expect egress. This method adds a defense-in-depth measure that protects against intentional or accidental misuse of the subnet.

Governance through policy

To align with your workload's security baseline, use Azure Policy and network policies to ensure that all workload resources meet your requirements. Platform automation through policy reduces the risk of security configuration drift and helps reduce manual validation activities.

Consider implementing the following types of security policies to strengthen your architecture:

- Disable key-based or other local authentication methods in services like Azure AI services and Key Vault.
- Require explicit configuration of network access rules, private endpoints, and NSGs.
- Require encryption, such as customer-managed keys.
- Restrict resource creation, such as limiting regions or resource types.

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

This [Azure pricing estimate](#) provides a pricing example for this architecture. This example includes only the components in this architecture, so customize it to match your usage. The most expensive components in the scenario are Azure Cosmos DB, AI Search, and DDoS Protection. Other notable costs include the chat UI compute and Application Gateway. Optimize those resources to reduce costs.

Agent Service

When you use the standard deployment, you must provision and manage the service's dependencies in your own Azure subscription.

The following recommendations explain how to optimize costs for these required services:

- Agent Service manages the request unit (RU) allocation on Azure Cosmos DB. To reduce long-term costs, purchase reserved capacity for Azure Cosmos DB. Align reservations with your expected usage duration and volume. Keep in mind that reserved capacity requires

upfront commitment and lacks flexibility if your workload's usage patterns change significantly.

- If your chat scenario doesn't require file uploads, exclude this feature in your application. In that case, apply the following configuration changes:
 - Use a locally redundant storage (LRS) tier for the Storage account.
 - Configure AI Search with a single replica instead of the recommended three replicas.
- Regularly delete unused agents and their associated threads by using the SDK or REST APIs. Stale agents and threads continue to consume storage and can increase costs across Azure Cosmos DB, Storage, and AI Search.
- Disable features on dependent resources that your workload doesn't require, such as the following features:
 - The semantic ranker in AI Search
 - The gateway and multi-region write capabilities in Azure Cosmos DB
- To avoid cross-region bandwidth charges, deploy Azure Cosmos DB, Storage, and AI Search in the same Azure region as Agent Service.
- Avoid colocating workload-specific data in the same Azure Cosmos DB or AI Search resources that Agent Service uses. In some cases, you can share these resources to reduce unused capacity in RUs or search units, which reduces cost. Only consider resource sharing after a thorough risk assessment for reliability, security, and performance trade-offs.

Agent knowledge and tools

Agent Service runs agent logic in a nondeterministic manner. It might invoke any connected knowledge store, tool, or other agent for each request, even if that resource isn't relevant to the user query. This behavior can result in unnecessary calls to external APIs or data sources, increase costs for each transaction, and introduce unpredictable usage patterns that complicate budget forecasting.

To control costs and maintain predictable behavior, apply the following strategies:

- Connect only the knowledge stores, tools, or agents that are likely to be used with most agent invocations. Avoid connecting resources that are rarely needed or that incur high costs for each call unless they're essential.
- Carefully design and refine the system prompt to instruct the agent to minimize unnecessary or redundant external calls. The system prompt should guide the agent to use only the most relevant connections for each request.

- Use Foundry telemetry to monitor which external APIs, knowledge stores, or tools the agent accesses, how frequently it accesses them, and the associated costs. Regularly review this telemetry to identify unexpected usage patterns or cost spikes, and adjust your system prompt as needed.
- Be aware that nondeterministic invocation can make cost forecasting difficult, especially when integrating with metered APIs. If you require predictable costs, consider self-hosting the orchestrator by using deterministic code.

Azure OpenAI models

Model deployments in Foundry use the MaaS approach. Costs depend primarily on usage or pre-provisioned allocation.

To control consumption model costs in this architecture, use a combination of the following approaches:

- **Control clients.** Client requests drive most costs in a consumption model, so controlling agent behavior is crucial.

To reduce unnecessary usage, take the following actions:

- Approve all model consumers. Don't expose models in a way that allows unrestricted access.
- Enforce token-limiting constraints such as `max_tokens` and `max_completions` through agent logic. This control is only available in self-hosted orchestration. Agent Service doesn't support this functionality.
- Optimize prompt input and response length. Longer prompts consume more tokens, which increases cost. Prompts that lack sufficient context reduce model effectiveness. Create concise prompts that provide enough context to allow the model to generate a useful response. Ensure that you optimize the limit of the response length.

This level of control is only available in self-hosted orchestration. Agent Service doesn't provide enough configuration to support this functionality.

- **Choose the right model for the agent.** Select the least expensive model that meets your agent's requirements. Avoid using higher cost models unless they're essential. For example, the reference implementation uses GPT-4o instead of a more expensive model and achieves sufficient results.
- **Monitor and manage usage.** Use [Microsoft Cost Management](#) and model telemetry to track token usage, set budgets, and create alerts for anomalies. Regularly review usage

patterns and adjust quotas or client access as needed. For more information, see [Plan and manage costs for Foundry](#).

- **Use the right deployment type.** Use pay-as-you-go pricing for unpredictable workloads, and switch to provisioned throughput when usage is stable and predictable. Combine both options when you establish a reliable baseline.
- **Restrict playground usage.** To prevent unplanned production costs, restrict the use of the Foundry playground to preproduction environments only.
- **Plan fine-tuning and image generation carefully.** These features have different billing models. They're billed per hour or per batch. Schedule usage to align with billing intervals and avoid waste.

Network security resources

This architecture requires Azure Firewall as an egress control point. To optimize costs, use the Basic tier of Azure Firewall unless the rest of your workload components require advanced features. Higher tiers add cost, so only use them if you need their capabilities.

If your organization uses an Azure landing zone, consider using shared firewall and distributed denial of service (DDoS) resources to defer or reduce costs. Workloads that have similar security and performance requirements can benefit from shared resources. Ensure that shared resources don't introduce security or operational risks. This architecture, [deployed in an Azure landing zone](#), uses shared resources.

Microsoft Defender for Cloud

This architecture should have the following Cloud Workload Protection plans enabled, and covering the related resources of the workload.

[] [Expand table](#)

| Plan | Benefit |
|--------------------------|---|
| Defender for Servers | Capabilities such as vulnerability assessments and file integrity monitoring helps prevent your highly privileged jump boxes and build agents role from becoming a threat vector. |
| Defender for App Service | Provides security monitoring of logs, host machines, and management interfaces for your chat interface components. |

| Plan | Benefit |
|------------------------------|---|
| Defender for Azure Cosmos DB | Provides database interaction monitoring for the database containing the chat threads, looking for signs of potential abuse or irregular access of your users' chat data and agent definitions. |
| Defender for AI services | Provides alerts based on your workload's requests and responses to your agents; alerting on attempts at jailbreaking or data leakage. If your organization uses Microsoft Purview, this plan also enables the licensed integration with Microsoft Purview DSPM for AI . |

If your organization hosts a security information and event management (SIEM) solution or uses Microsoft Purview, ensure that any customer data, such as prompts and responses, that's replicated into their data stores resides in a region that doesn't violate any data sovereignty restrictions your workload requires.

Operational Excellence

Operational Excellence covers the operations processes that deploy an application and keep it running in production. For more information, see [Design review checklist for Operational Excellence](#).

The following operational excellence guidance doesn't include the front-end application elements, which remain the same as the [baseline highly available zone-redundant web application architecture](#).

When planning your experimentation, testing, and production environments, establish distinct and isolated Foundry resources, including dependencies.

Agent compute

Microsoft fully manages the serverless compute platform for Azure AI Agent REST APIs and the orchestration implementation logic. A [self-hosted orchestration](#) provides more control over runtime characteristics and capacity, but you must directly manage the day-2 operations for that platform. Evaluate the constraints and responsibilities of your approach to understand which day-2 operations you must implement to support your orchestration layer.

In both approaches, you must manage state storage, such as chat history and agent configuration for durability, backup, and recovery.

Monitoring

Similar to the basic architecture, diagnostics data from all services is configured to be sent to your workload's Log Analytics workspace. All services except App Service capture all logs. In production, you might not need to capture all logs. For example, the chat UI application might only require `AppServiceHTTPLogs`, `AppServiceConsoleLogs`, `AppServiceAppLogs`, `AppServicePlatformLogs`, and `AppServiceAuthenticationLogs`. Tune log streams to your operational needs.

Evaluate custom alerts, such as custom alerts in the Azure Monitor baseline alerts, for the resources in this architecture. Consider the following alerts:

- [AI Search alerts ↗](#)
- [AI services alerts ↗](#)
- [Web Apps alerts ↗](#)

Monitor the usage of tokens against your model deployments. In this architecture, Foundry tracks [token usage](#) through its integration with Application Insights.

Your jump boxes and build agent VMs reside in a highly privileged location, which provides those VMs a network line of sight to the data plane of all components in your architecture. Ensure that those VMs emit enough logs to understand when they're used, who uses them, and what actions they perform.

Agent versioning and life cycle

Treat each agent as an independently deployable unit within your chat workload, unless you specifically design your application to dynamically create and delete agents at runtime. These agents have life cycle management requirements similar to other microservices in your workload.

To prevent service disruptions, ensure safe and controlled agent deployment by implementing the following approaches:

- **Define agents as code.** Always store agent definitions, connections, system prompts, and configuration parameters in source control. This practice ensures traceability and reproducibility. Avoid untracked changes through the Foundry portal.
- **Automate agent deployment.** Use your workload's continuous integration and continuous deployment (CI/CD) pipelines. Use the Azure AI Agent SDK to build, test, and deploy agent changes from your network-connected build agents.

Prefer agent pipelines that you can deploy independently for small, incremental changes. But make sure that the pipelines provide enough flexibility to deploy them alongside your application code when you require coordinated updates. To support this method, loosely

couple your chat UI code and your chat agents so that changes to one component don't require simultaneous changes to the other.

- **Test before production.** Validate agent behavior, prompts, and connections in preproduction environments. Use a combination of automated and manual tests to catch regressions, security problems, and unintended changes in agent behavior.

Agents defined in Agent Service behave nondeterministically, so you must determine how to measure and maintain your desired quality level. Create and run a test suite that checks for ideal responses to realistic user questions and scenarios.

- **Version and track agents.** Assign clear version identifiers to each agent. Maintain records of which agent versions are active, along with their dependencies such as models, data sources, and tools. Prefer deploying new agent versions alongside existing ones to enable progressive rollout, rollback, and controlled migration of users or sessions.
- **Plan for fallback.** Foundry doesn't provide built-in support for blue-green or canary deployments of agents. If you require these deployment patterns, implement a routing layer, such as an API gateway or custom router, in front of the agent API. This routing layer allows you to shift traffic incrementally between agent versions, monitor the effect, and perform a full switchover when ready.
- **Coordinate agent removal.** When you remove agents, coordinate the process with your application's state management and user experience requirements. Handle active chat sessions appropriately. Depending on your workload's functional requirements, you can migrate sessions, pin users to the old agent version, or require users to start new sessions.

Performance Efficiency

Performance Efficiency refers to your workload's ability to scale to meet user demands efficiently. For more information, see [Design review checklist for Performance Efficiency](#).

This section addresses performance efficiency for AI Search, model deployments, and Foundry.

Performance Efficiency in AI Search

In Agent Service, you don't control the specific queries sent to your indexes because agents are codeless. To optimize performance, focus on what you can control in the index. Observe how your agent typically queries the index, and apply guidance to [analyze and optimize performance in AI Search](#).

If index server-tuning alone doesn't resolve all bottlenecks, consider the following options:

- Replace the direct connection to AI Search with a connection to an API that you own. This API can implement code optimized for your agent's retrieval patterns.
- Redesign the orchestration layer to use the [self-hosted alternative](#) so that you can define and optimize queries in your own orchestrator code.

Performance Efficiency in model deployments

- Determine whether your application needs [provisioned throughput](#) or can use the shared (consumption) model. Provisioned throughput provides reserved capacity and predictable latency, which is important for production workloads that have strict service-level objectives. The consumption model provides best-effort service and might suffer from noisy neighbor effects.
- Monitor [provision-managed usage](#) to avoid overprovisioning or underprovisioning.
- Choose a conversational model that meets your inference latency requirements.
- Deploy models in the same data region as your agents to minimize network latency.

Azure AI agent performance

Azure AI agents run on a serverless compute back end that doesn't support custom performance tuning. However, you can still improve performance through agent design:

- Minimize the number of knowledge stores and tools connected to your chat agent. Each extra connection potentially increases the total runtime for an agent call because the agent might invoke all configured resources for each request.
- Use Azure Monitor and Application Insights to track agent invocation times, tool and knowledge store latencies, and error rates. Regularly review this telemetry to identify slow knowledge or tool connections.
- Design system prompts that guide the agent to use connections efficiently. For example, instruct the agent to query external knowledge stores only when needed, or to avoid redundant tool invocations.
- Monitor for service limits or quotas that might affect performance during peak usage. Watch for throttling indicators such as HTTP 429 or 503 responses, even though serverless compute scales automatically.

Deploy this scenario

To deploy and run this reference implementation, follow the deployment guide in the [Agent Service chat baseline reference implementation](#).

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal authors:

- [Rob Bagby](#) | Principal Content Developer - Azure Patterns & Practices
- [Chad Kittel](#) | Principal Software Engineer - Azure Patterns & Practices

Other contributors:

- [Raouf Aliouat](#) | Senior Software Engineer
- [Freddy Ayala](#) | Cloud Solution Architect
- [Prabal Deb](#) | Principal Software Engineer
- [Ritesh Modi](#) | Principal Software Engineer
- [Ryan Pfalz](#) | Senior Technical Program Manager

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next step

- [Baseline Foundry chat architecture in an Azure landing zone](#)

Related resources

- An Azure Well-Architected Framework perspective on [AI workloads on Azure](#)
- [Azure OpenAI models](#)
- [Content filtering](#)
- [AI agent orchestration patterns](#)

Baseline Microsoft Foundry chat reference architecture in an Azure landing zone

Azure OpenAI Service

Azure AI services

Azure App Service

Azure Key Vault

Azure Monitor

Azure AI Foundry

This article is part of a series that builds on the [Baseline Microsoft Foundry chat reference architecture](#). Review the baseline architecture so that you can identify necessary adjustments before you deploy it in an Azure application landing zone subscription.

This article describes a generative AI workload architecture that deploys the baseline chat application but uses resources that are outside the workload team's scope. Platform teams centrally manage the resources, and multiple workload teams use them. Shared resources include networking resources for cross-premises connections, identity access management systems, and policies. This guidance helps organizations that use Azure landing zones maintain consistent governance and cost efficiency.

Foundry uses resources and projects to organize AI development and deployment. For example, a landing zone implementation might use a Foundry resource as a centralized resource at a business group level and projects as a delegated resource for each workload in that business group. Because of resource organization factors, and cost allocation limitations, we don't recommend this topology, and this article doesn't provide guidance about it. Instead, this architecture treats the workload as the owner of the Foundry resource, which is the recommended approach.

As a workload owner, you delegate shared resource management to platform teams so that you can focus on workload development efforts. This article presents the workload team's perspective and specifies recommendations for the platform team.

Important

What are Azure landing zones?

Azure landing zones divide your organization's cloud footprint into two key areas:

- An application landing zone is one or more Azure subscriptions where a workload runs. An application landing zone connects to your organization's shared platform resources. That connection provides the landing zone with access to the infrastructure that supports the workload, such as networking, identity access management, policies, and monitoring.

- A platform landing zone is a collection of various subscriptions that multiple platform teams can manage. Each subscription has a specific function. For example, a connectivity subscription provides centralized Domain Name System (DNS) resolution, cross-premises connectivity, and network virtual appliances (NVAs) for platform teams.

To help you implement this architecture, understand [Azure landing zones](#), their [design principles](#), and their [design areas](#).

Article layout

 Expand table

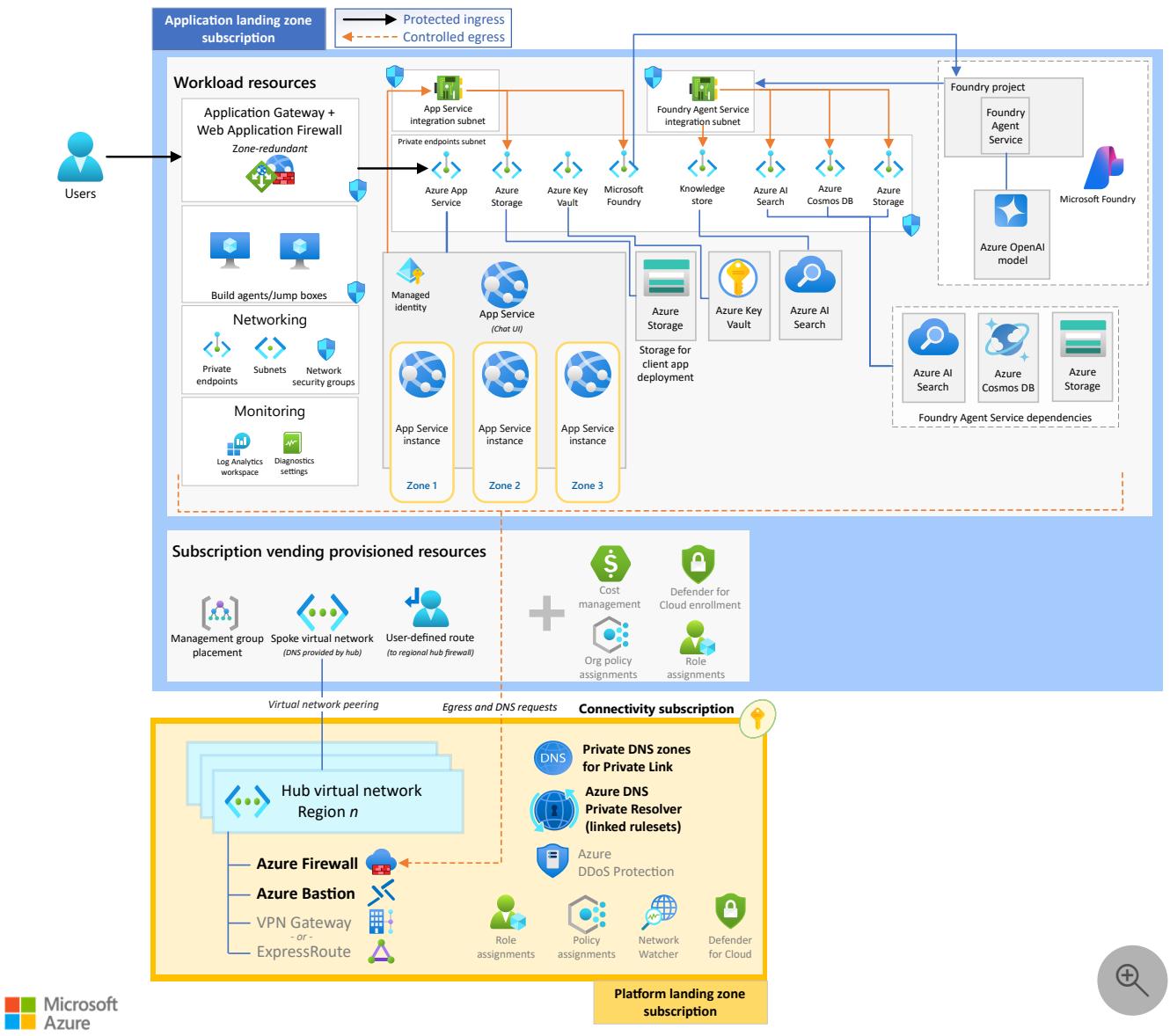
| Architecture | Design decisions | Azure Well-Architected Framework approach |
|---|---|--|
| <ul style="list-style-type: none">▪ Architecture diagram▪ Workload resources▪ Federated resources | <ul style="list-style-type: none">▪ Subscription setup▪ Networking▪ Data scientist access▪ Monitor resources▪ Organizational governance▪ Change management | <ul style="list-style-type: none">▪ Reliability▪ Security▪ Cost Optimization▪ Operational Excellence▪ Performance Efficiency |

Tip



The [Foundry Agent Service chat baseline reference implementation](#) ↗ demonstrates the best practices described in this article. Review and try these deployment resources before you choose and implement your design decisions.

Architecture



Download a [Visio file](#) of this architecture.

Components

All Azure landing zone architectures separate ownership between the platform team and the workload team, which is referred to as **subscription democratization**. Application architects, data scientists, and DevOps teams must clearly understand this division to determine what falls under their direct influence or control and what doesn't.

Like most application landing zone implementations, the workload team primarily manages the configuration, deployment, and oversight of workload components, including the AI services in this architecture.

Workload team-owned resources

The following resources remain mostly unchanged from the **baseline architecture**.

- **Foundry resource** and **projects** is an application platform for AI developers and data scientists to build, evaluate, and deploy AI models and host agents. In this architecture, the Foundry resource enables the workload team to host generative AI models as a service (MaaS), implement content safety, and establish workload-specific connections to knowledge sources and tools.

If your organization's AI Center of Excellence restricts access to AI model deployments, the workload team might not host models in their own Foundry resource. Instead, they might need to use **centralized AI resources** such as an AI hub. In this scenario, all model consumption usually flows through an AI gateway that your AI platform team provides.

This article assumes that generative AI models in this scenario are workload-owned resources. If they're not, the model host or an **AI gateway** becomes a workload dependency. The platform team must maintain reliable network connectivity from your virtual network to their virtual network or a private endpoint must be established.

- **Agent Service** is a cloud-native runtime environment that enables intelligent agents to operate securely and autonomously. In this architecture, Agent Service provides the orchestration layer for chat interactions. It hosts and manages the chat agent that processes user requests. This architecture supports both declarative and containerized agents.

Use the **standard agent setup** in this architecture. Connect your agent to a dedicated subnet in your spoke virtual network, and route egress traffic through your connectivity subscription.

The workload team supplies dedicated Azure resources for agent state, chat history, and file storage. These resources are **Azure Cosmos DB for NoSQL**, **Azure Storage**, and **Azure AI Search**. Your Agent Service instance manages these resources and their data exclusively. Other application components in your workload or other workloads in your organization shouldn't use them.

- **Azure App Service** enables developers to build web and mobile apps and automate business processes with API apps. In this architecture, it hosts the web application that contains the chat user interface (UI) and delivers the user-facing component, with multiple instances across zones for availability.

An Azure Storage account hosts the web application's code as a ZIP file, which mounts within App Service.

- **AI Search** is a scalable search infrastructure that indexes heterogeneous content and enables data retrieval through APIs, applications, and AI agents. In this architecture, Foundry IQ powered by AI Search serves as the workload knowledge store for the **Retrieval Augmented Generation pattern**. This pattern extracts an appropriate query from

a prompt, queries AI Search, and uses the results as grounding data for a generative AI foundation model.

- **Azure Application Gateway** is a web traffic load balancer and application delivery controller. In this architecture, it acts as the reverse proxy to route user requests to the chat UI hosted in App Service. It hosts an Azure web application firewall to help protect the front-end application from potentially malicious traffic.
- **Azure Key Vault** is a cloud service for securely storing and accessing secrets, keys, and certificates. In this architecture, it stores the application gateway's Transport Layer Security (TLS) certificate.
- **Azure Monitor**, **Azure Monitor Logs**, and **Application Insights** collect, store, and visualize observability data. In this architecture, they enable monitoring, diagnostics, and operational insights for all workload components.
- **Azure Policy** applies workload-specific policies to help govern, secure, and apply controls at scale. In this architecture, it enforces governance and compliance rules on resources that the workload team manages.

The workload team also maintains the following resources:

- **Spoke virtual network subnets and the network security groups (NSGs)** maintain segmentation and control traffic flow between subnets. In this architecture, they enforce network boundaries and security between workload components.
- **Private endpoints** secure connectivity to platform as a service (PaaS) solutions. In this architecture, they ensure that sensitive services are only accessible within the private network, which reduces exposure to the public internet. Additional dependencies, such as state stores owned by other team, can be exposed to your workload as a private endpoint, avoiding transitive routing through the connectivity subscription.

Platform team-owned resources

The platform team owns and maintains the following centralized resources. This architecture assumes that these resources are pre-provisioned and treats them as dependencies.

- **Azure Firewall** is a managed, cloud-based network security service. In this architecture, Azure Firewall in the hub network routes, inspects, and restricts egress traffic that originates from the workload, including agent traffic. Workload egress traffic goes to the internet, cross-premises destinations, or to other application landing zones.

Change from the baseline: In the baseline architecture, the workload team owns this component. In this architecture, the platform team manages it under the connectivity

subscription.

- **Azure Bastion** is a managed PaaS service that you use to connect to virtual machines via private IP address. In this architecture, Azure Bastion in the hub network provides secure operational access to workload components and allows access to Foundry components. It ensures that administrators and support staff can securely connect to virtual machines without exposing RDP/SSH ports to the public internet.

Change from the baseline: In the baseline architecture, the workload team owns this component.

- The **spoke virtual network** is where the workload is deployed.

Change from the baseline: In the baseline architecture, the workload team owns this network.

- **User-defined routes (UDRs)** enforce tunneling to the hub network's firewall.

Change from the baseline: In the baseline architecture, the workload team owns this routing.

- **Azure Policy-based governance constraints** and `DeployIfNotExists` (DINE) policies apply to the workload subscription. You can apply these policies at the platform team-owned management group level or to the workload's subscription directly.

Change from the baseline: These policies are new in this architecture. The platform team applies policies that constrain your workload. Some policies might duplicate existing workload constraints or introduce new constraints.

- **Azure private DNS zones** host `A` records for private endpoints so that workload components can securely resolve the FQDN of PaaS services used within the workload. For more information, see [Azure Private Link and DNS integration at scale](#).

Change from the baseline: In the baseline architecture, the workload team owns this network. In this architecture, the platform team manages this component under the connectivity subscription.

- **DNS resolution service** supports spoke virtual networks and cross-premises workstations. This service typically uses Azure Firewall as a DNS proxy or Azure DNS Private Resolver. In this architecture, the service resolves private endpoint DNS records for all DNS requests from the spoke. DNS Private Resolver and linked rulesets is the recommended way for the platform team to enable this architecture resolution requirements due to the DNS resolution characteristics of Agent Service.

- **Azure DDoS Protection** helps protect public IP addresses from distributed attacks. In this architecture, DDoS Protection helps safeguard the public IP address that's associated with Application Gateway.

Change from the baseline: In the baseline architecture, the workload team purchases DDoS Protection.

Important

Azure landing zones provide some of the preceding resources as part of the platform landing zone subscriptions. Your workload subscription provides other resources. Many of these resources reside in the connectivity subscription, which also includes Azure ExpressRoute, Azure VPN Gateway, and DNS Private Resolver. These resources provide cross-premises access and name resolution. The management of these resources falls outside the scope of this article.

Subscription setup

The workload team must inform the platform team of specific landing zone requirements to implement this architecture. And the platform team must communicate its requirements to the workload team.

For example, the workload team must provide detailed information about the required networking space. The platform team uses this information to allocate the necessary resources. The workload team defines the requirements, and the platform team assigns the appropriate IP addresses within the virtual network.

The platform team assigns a management group based on the workload's business criticality and technical needs. For instance, if the workload is exposed to the internet, like this architecture, the platform team selects an appropriate management group. To establish governance, the platform team also configures and implements management groups. The workload team must design and operate the workload within the constraints of this governance. For more information about typical management group distinctions, see [Tailor the Azure landing zone architecture](#).

The platform team sets up the subscription for this architecture. The following sections provide guidance about the initial subscription setup.

Workload requirements and fulfillment

The workload team and platform team must collaborate on details like management group assignment, Azure Policy governance, and networking setup. Prepare a checklist of requirements to initiate discussion and negotiation with the platform team. The following checklist serves as an example.

[+] Expand table

| Design consideration | Workload requirement for this architecture |
|--|--|
| <ul style="list-style-type: none"><input type="checkbox"/> The number of spoke virtual networks and their size: The platform team creates and configures the virtual network, then peers it to the regional hub to designate it as a spoke. They also need to ensure that the network can accommodate future workload growth. To carry out these tasks effectively, they must know the number of spokes required. | <p>Deploy all resources in a single, dedicated spoke virtual network. Request /22 contiguous address space to support full-scale operations and scenarios like side-by-side deployments.</p> <p>The following factors determine most IP address needs:</p> <ul style="list-style-type: none">- Application Gateway requirements for the subnet size (fixed size).- Private endpoints with single IP addresses for PaaS services (fixed size).- The subnet size for build agents (fixed size).- Agent Service requires a subnet within a /24 prefix. |
| <ul style="list-style-type: none"><input type="checkbox"/> Virtual network address prefixes: Typically, the platform team assigns IP addresses based on existing conventions, avoidance of overlap with peered networks, and availability within the IP address management (IPAM) system. | <p>The agent integration subnet must use a valid private address prefix. Agent Service supports subnets that use 10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16. Ask your platform team to provide a spoke that has a valid address prefix for your agent subnet.</p> |
| <ul style="list-style-type: none"><input type="checkbox"/> Deployment region: The platform team needs to deploy a hub in the same region as the workload resources. | <p>Communicate the selected region for the workload and the regions for underlying compute resources. Ensure that the regions support availability</p> |

| Design consideration | Workload requirement for this architecture |
|---|---|
| | <p>zones. Azure OpenAI in Foundry Models has limited regional availability.</p> |
| <ul style="list-style-type: none"> <input type="checkbox"/> Data sovereignty: The platform team needs to honor all data residency requirements the workload has. | <p>If your workload requires strict data residency, ensure the platform team isn't duplicating Azure Diagnostics logs or sending data to Purview in a region not allowed by your requirements.</p> |
| <ul style="list-style-type: none"> <input type="checkbox"/> Type, volume, and pattern of traffic: The platform team needs to determine the ingress and egress requirements of your workload's shared resources. | <p>Provide information about the following factors:</p> <ul style="list-style-type: none"> - How users should consume this workload. - How this workload consumes its surrounding resources. - The configured transport protocol. - The traffic pattern and the expected peak and off-peak hours. Communicate when you expect a high number of concurrent connections to the internet (chatty) and when you expect the workload to generate minimal network traffic (background noise). |
| <ul style="list-style-type: none"> <input type="checkbox"/> Firewall configuration: The platform team needs to set rules to allow legitimate egress traffic. | <p>Share details about outbound traffic from the spoke network, including agent traffic.</p> <p>Build agent and jump box machines need regular OS patching.</p> <p>Agents might need to interact with internet grounding sources, tools, or other agents hosted outside the workload.</p> |

| Design consideration | Workload requirement for this architecture |
|--|--|
| <ul style="list-style-type: none"> <input type="checkbox"/> Ingress traffic from specialized roles: The platform team needs to provide the specified roles with network access to the workload and implement proper segmentation. | <p>Work with the platform team to determine the best way to allow authorized access for the following roles:</p> <ul style="list-style-type: none"> - Data scientists and developers that access the Foundry portal from their workstations on corporate network connections - Operators that access the compute layer through a workload-managed jump box |
| <ul style="list-style-type: none"> <input type="checkbox"/> Public internet access to the workload: The platform team uses this information for risk assessment, which drives several decisions: <ul style="list-style-type: none"> - The placement of the workload in a management group with appropriate guardrails - Distributed denial-of-service (DDoS) protection for the public IP address reported by the workload team - TLS certificate procurement and management | <p>Inform the platform team about the ingress traffic profile:</p> <ul style="list-style-type: none"> - Internet-sourced traffic that targets the public IP address on Application Gateway - Fully qualified domain names (FQDNs) associated with the public IP address for TLS certificate procurement |
| <ul style="list-style-type: none"> <input type="checkbox"/> Private endpoint usage: The platform team needs to set up Azure private DNS zones for the private endpoints and ensure that the firewall in the hub network performs DNS resolution correctly. | <p>Understand how the hub handles DNS resolution, and define the workload team's responsibilities for the management of private DNS zone records and DNS Private Resolver ruleset linking.</p> <p>Inform the platform team about all resources that use private endpoints, including the following resources:</p> <ul style="list-style-type: none"> - AI search - Azure Cosmos DB for NoSQL - Key Vault - Foundry - Storage accounts |

| Design consideration | Workload requirement for this architecture |
|---|---|
| <ul style="list-style-type: none"> <input type="checkbox"/> Centralized AI resources: The platform team must understand the expected usage of models and hosting platforms. They use this information to establish networking to centralized AI resources within your organization. <p>Each organization defines its own AI adoption and governance plans, and the workload team must operate within those constraints.</p> | <p>Inform the platform team about AI and machine learning resources that you plan to use. This architecture uses Foundry, Agent Service, and generative foundation models hosted in Foundry Models.</p> <p>Clearly understand which centralized AI services you must use and how those dependencies affect your workload.</p> |

i Important

The platform team should follow a subscription vending process that uses a structured set of questions to collect information from the workload team. These questions might vary across organizations, but the goal is to gather the necessary input to implement subscriptions effectively. For more information, see [Subscription vending](#).

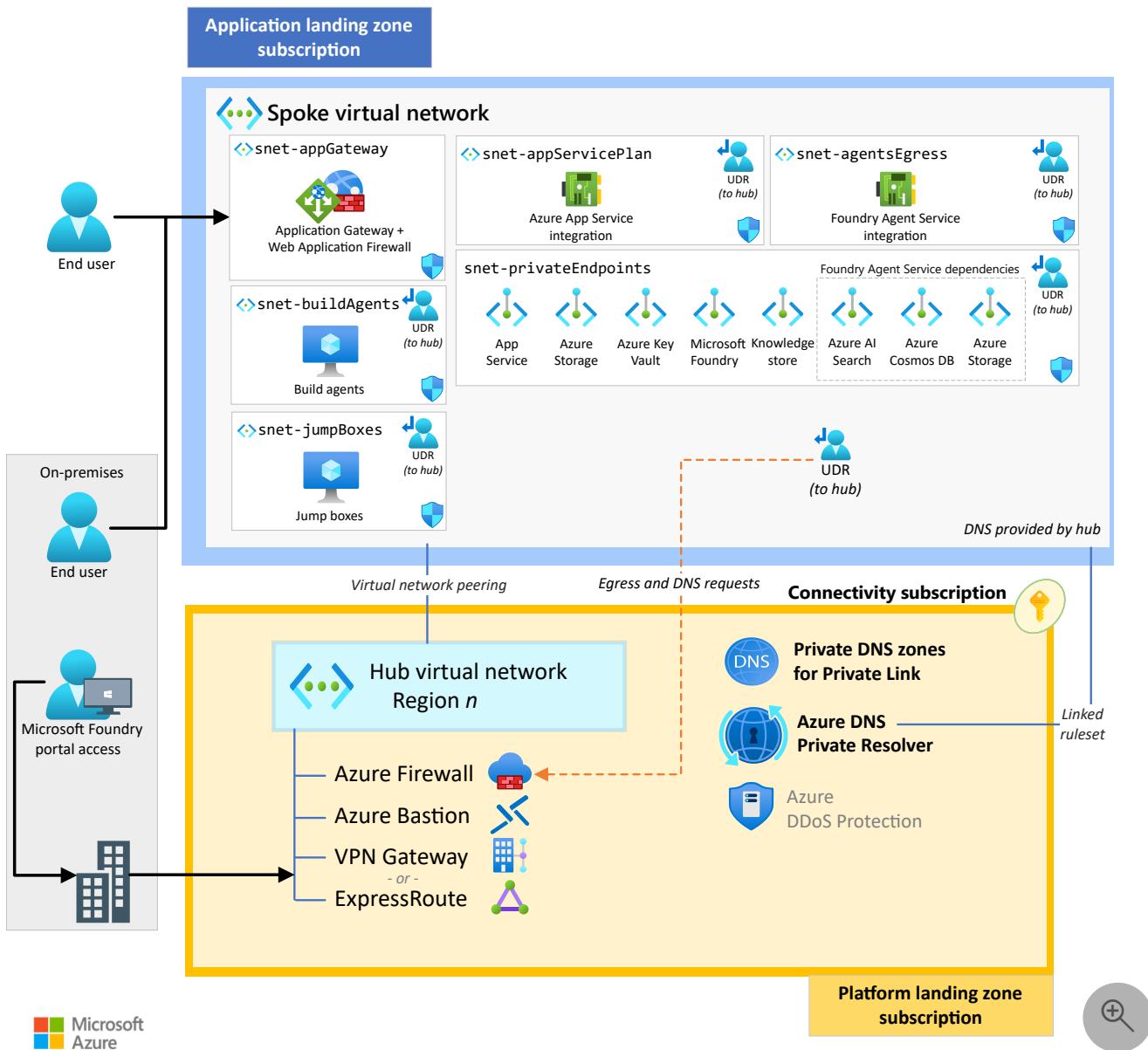
Compute

The orchestration layer and chat UI hosting remain the same as the [baseline architecture](#).

Networking

In the [baseline architecture](#), the workload is provisioned in a single virtual network.

Change from the baseline: This architecture divides the workload over two virtual networks. One network hosts workload components. The other network manages internet and hybrid connectivity. The platform team determines how the workload's virtual network integrates with the organization's larger network architecture, which typically follows a hub-spoke topology.



Download a [Visio file](#) of this architecture.

- **Hub virtual network:** This virtual network serves as a regional hub that contains centralized, and often shared, services that communicate with workload resources in the same region. The hub resides in the **connectivity subscription**. The platform team owns the resources in this network.
- **Spoke virtual network:** In this architecture, the single virtual network from the baseline architecture essentially becomes the spoke virtual network. The platform team peers this spoke network to the hub network. They own and manage the spoke network, including its peering and DNS configuration. The workload team owns the resources in this network, including its subnets. This network contains many of the **workload resources**.

Because of this division of management and ownership, the workload team must clearly communicate the **workload's requirements** to the platform team.

Important

For the platform team: Don't directly peer the spoke network to another spoke network, unless the workload specifically requires it. This practice protects the segmentation goals of the workload. Your team must facilitate all transitive virtual network connections. However, if application landing zone teams directly connect their networks by using self-managed private endpoints, your team doesn't manage those connections.

Understand which workload resources external teams manage. For example, understand the network connectivity between the chat agents and a grounding context vector database that another team manages.

Virtual network subnets

In the spoke virtual network, you create and allocate the subnets based on the workload requirements. To provide segmentation, apply controls that restrict traffic into and out of the subnets. This architecture doesn't add subnets beyond the [subnets in the baseline architecture](#). However, the network architecture no longer requires the `AzureBastionSubnet` or `AzureFirewallSubnet` subnets because the platform team likely hosts this capability in their subscriptions.

You still have to implement local network controls when you deploy your workload in an Azure landing zone. Your organization might impose further network restrictions to safeguard against data exfiltration and ensure visibility for the central security operations center and the IT network team.

Ingress traffic

The [ingress traffic flow remains the same as the baseline architecture](#).

You manage resources related to public internet ingress into the workload. For example, in this architecture, Application Gateway and its public IP address reside in the spoke network rather than the hub network. Some organizations place ingress-facing resources in a connectivity subscription by using a centralized perimeter network (also known as DMZ, demilitarized zone, and screened subnet) implementation. Integration with that specific topology falls outside the scope of this article.

Alternate approach to inspecting incoming traffic

This architecture doesn't use Azure Firewall to inspect incoming traffic, but sometimes organizational governance requires it. In those cases, the platform team supports the implementation to provide workload teams with an extra layer of intrusion detection and prevention. This layer helps block unwanted inbound traffic. To support this topology, this architecture requires more UDR configurations. For more information, see [Zero Trust network for web applications with Azure Firewall and Application Gateway](#).

DNS configuration

In the baseline architecture, all components use Azure DNS directly for DNS resolution.

Change from the baseline: In this architecture, typically one or more DNS servers in the hub perform DNS resolution. When the virtual network is created, the DNS properties on the virtual network should already be set accordingly. The workload team doesn't need to understand the implementation details of the DNS service.

This architecture configures the workload components with DNS in the following ways.

[] [Expand table](#)

| Component | DNS configuration |
|-----------------------|-------------------------------------|
| Application Gateway | Inherited from virtual network |
| App Service (chat UI) | Inherited from virtual network |
| AI Search | Can't be overridden, uses Azure DNS |
| Foundry | Can't be overridden, uses Azure DNS |
| Agent Service | Can't be overridden, uses Azure DNS |
| Azure Cosmos DB | Can't be overridden, uses Azure DNS |
| Jump box | Inherited from virtual network |
| Build agents | Inherited from virtual network |

This architecture doesn't configure DNS settings for components that don't initiate outbound communication. These components don't require DNS resolution.

Many components in this architecture rely on DNS records hosted in the hub's DNS servers to resolve this workload's private endpoints. For more information, see [Azure private DNS zones](#).

When hub-based DNS resolution isn't possible, those components face the following limitations:

- The platform team can't log DNS requests, which might violate an organizational security team requirement.
- Resolving to Private Link-exposed services in your landing zone or other application landing zones might be impossible.

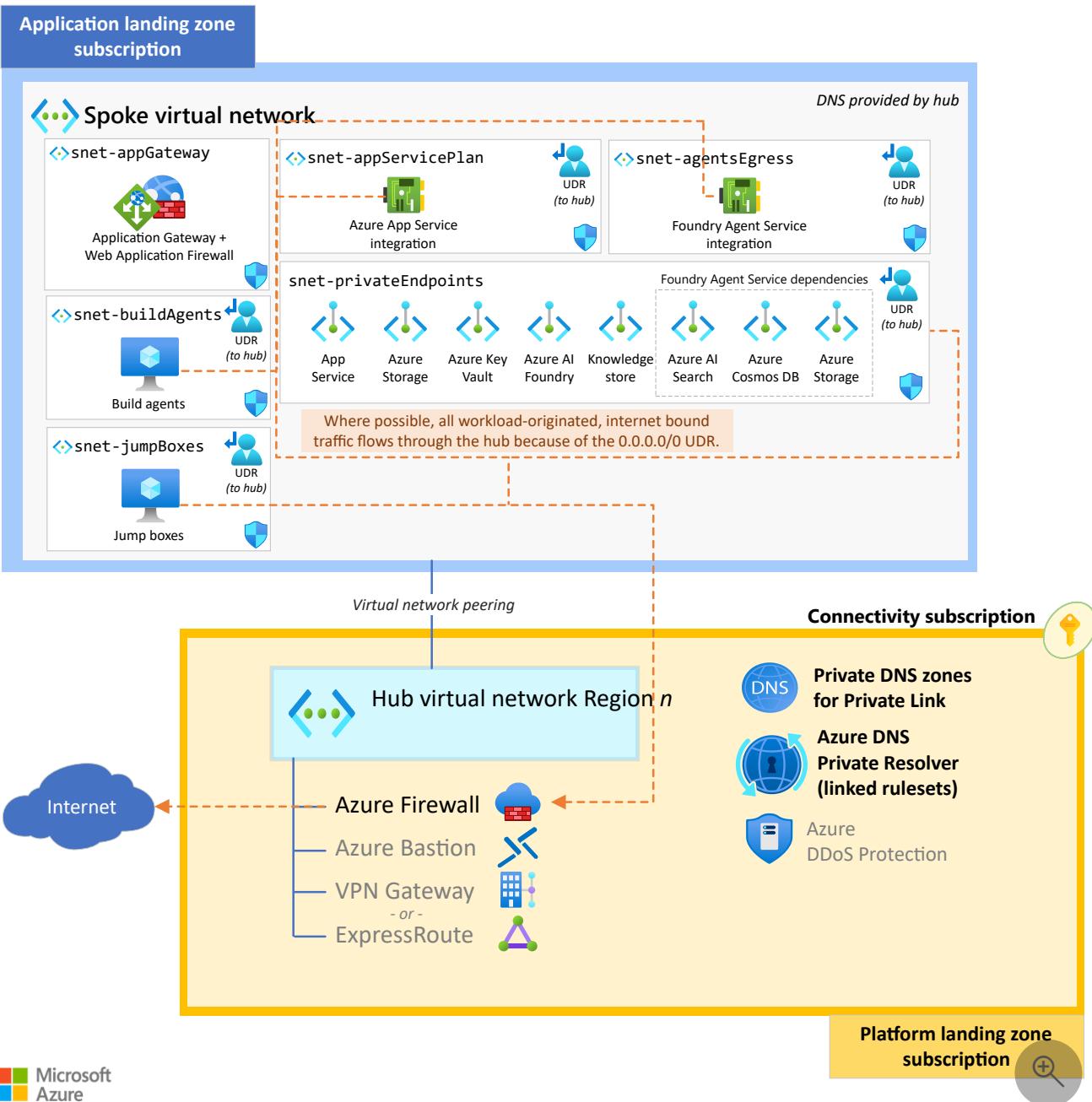
We recommend that you familiarize yourself with how the platform team manages DNS. For more information, see [Private Link and DNS integration at scale](#). When you add component features that directly depend on Azure DNS, you might introduce complexities in the platform-provided DNS topology. You can redesign components or negotiate exceptions to minimize complexity.

Egress traffic

In the baseline architecture, all egress traffic routes to the internet through Azure Firewall.

Change from the baseline: In this architecture, the platform provides a UDR that points to an Azure Firewall instance that it hosts. Apply this UDR to the same subnets in the baseline architecture.

All traffic that leaves the spoke virtual network, including traffic from the agent integration subnet, reroutes through the peered hub network via an egress firewall.



Download a [Visio file](#) of this architecture.

East-west client communication to the private endpoints for Key Vault, Foundry, and other services remains the same as the [baseline architecture](#). The preceding diagram doesn't include that path.

Route internet traffic to the firewall

All subnets in the spoke network include a route that directs all internet-bound traffic, or **0.0.0.0/0** traffic, to the hub's Azure Firewall instance.

 [Expand table](#)

| Component | Mechanism to force internet traffic through the hub |
|-----------------------|---|
| Application Gateway | None. Internet-bound traffic that originates from this service can't be forced through the platform team's firewall. |
| App Service (chat UI) | Regional virtual network integration and the <code>vnetRouteAllEnabled</code> setting are enabled. |
| AI Search | None. Traffic that originates from this service can't be forced through a firewall. This architecture doesn't use skills. |
| Agent Service | A UDR applied to the snet-agentsEgress subnet. |
| Jump boxes | A UDR applied to the snet-jumpbox subnet. |
| Build agents | A UDR applied to the snet-agents subnet. |

This architecture doesn't configure force tunneling for components that don't initiate outbound communication.

For components or features that can't route egress traffic through the hub, your workload team must align with organizational requirements. To meet those requirements, use compensating controls, redesign the workload to exclude unsupported features, or request formal exceptions. You're responsible for mitigating data exfiltration and abuse.

Apply the platform-provided internet route to all subnets, even if you don't expect the subnet to have outgoing traffic. This approach ensures that unexpected deployments in that subnet go through routine egress filtering. For subnets that contain private endpoints, enable network policies to support full routing and NSG control.

This route configuration ensures that all outbound connections from App Service, Foundry and its projects, and any other services that originate from the workload's virtual network are inspected and controlled.

Azure private DNS zones

Workloads that use private endpoints for east-west traffic require DNS zone records in the configured DNS provider. To support Private Link, this architecture relies on many DNS zone records for services such as Key Vault, Foundry, and Azure Storage.

Change from the baseline: In the baseline architecture, the workload team directly manages the private DNS zones. In this architecture, the platform team typically maintains private DNS zones. The workload team must clearly understand the platform team's requirements and expectations for the management of the private DNS zone records. The platform team can use other technology instead of private DNS zone records.

In this architecture, the platform team must set up DNS for the following Foundry FQDN API endpoints:

- privatelink.services.ai.azure.com
- privatelink.openai.azure.com
- privatelink.cognitiveservices.azure.com

The platform team must also set up DNS for the following FQDNs, which are Agent Service dependencies:

- privatelink.search.windows.net
- privatelink.blob.core.windows.net
- privatelink.documents.azure.com

Important

DNS resolution must function correctly from within the spoke virtual before you deploy the capability host for Agent Service and during operation of the Agent Service. The Agent Service capability doesn't use your spoke virtual network's DNS configuration. Therefore, it's recommended that your platform team configure a ruleset for the workload's private DNS zones in DNS Private Resolver, linking those rules to your application landing zone spoke.

Before you deploy Foundry and its agent capability, you must wait until the Agent Service dependencies are fully resolvable to their private endpoints from within the spoke network. This requirement is especially important if DINE policies handle updates to DNS private zones. If you attempt to deploy the Agent Service capability before the private DNS records are resolvable from within your subnet, the deployment fails.

The platform team must also host the private DNS zones for other workload dependencies in this architecture:

- privatelink.vaultcore.azure.net
- privatelink.azurewebsites.net

Data scientist and agent developer access

Like the [baseline architecture](#), this architecture disables public ingress access to the Foundry portal and other browser-based experiences. The baseline architecture deploys a jump box to provide a browser with a source IP address from the virtual network that various workload roles use.

When your workload connects to an Azure landing zone, your team gains more access options. Work with the platform team to see if you can get private access to various browser-based Foundry portals without managing and governing a virtual machine (VM). This access might be possible through transitive routing from an existing ExpressRoute or VPN Gateway connection.

Native workstation-based access requires cross-premises routing and DNS resolution, which the platform team can help provide. Include this requirement in your subscription vending request.

Providing native workstation-based access to these portals improves productivity and simplifies maintenance compared to managing VM jump boxes.

The role of the jump box

The jump box in this architecture provides value for operational support, not for runtime purposes or AI and machine learning development. The jump box can troubleshoot DNS and network routing problems because it provides internal network access to otherwise externally inaccessible components.

In the baseline architecture, Azure Bastion accesses the jump box, which you manage.

In this architecture, Azure Bastion is deployed in the connectivity subscription as a shared regional resource that the platform team manages. To demonstrate that use case in this architecture, Azure Bastion is in the connectivity subscription and your team doesn't deploy it.

The VM that serves as the jump box must comply with organizational requirements for VMs. These requirements might include items such as corporate identities in Microsoft Entra ID, specific base images, and patching regimes.

Monitor resources

The Azure landing zone platform provides shared observability resources as part of the management subscription. However, we recommend that you provision your own monitoring resources to facilitate ownership responsibilities of the workload. This approach aligns with the [baseline architecture](#).

You provision the following monitoring resources:

- Application Insights serves as the application performance management (APM) service for your team. You configure this service in the chat UI, Agent Service, and models.
- The Azure Monitor Logs workspace serves as the unified sink for all logs and metrics from workload-owned Azure resources.

Similar to the baseline architecture, all resources must send Azure diagnostics logs to the Azure Monitor Logs workspace that your team provisions. This configuration is part of the infrastructure as code (IaC) deployment of the resources. You might also need to send logs to a central Azure Monitor Logs workspace. In Azure landing zones, that workspace typically resides in the management subscription.

The platform team might have more processes that affect resources in the application landing zone. For example, they might use DINE policies to configure diagnostics and send logs to centralized management subscriptions. They might also apply [Azure Monitor baseline alerts](#) through policy. Ensure that your implementation doesn't block these extra logging and alerting flows.

Azure Policy

The baseline architecture recommends [general policies](#) to help govern the workload. When you deploy this architecture into an application landing zone, you don't need to add or remove extra policies. To help enforce governance and enhance the security of this workload, continue to apply policies to your subscription, resource groups, or resources.

Expect the application landing zone subscription to have existing policies, even before you deploy the workload. Some policies help organizational governance by auditing or blocking specific configurations in deployments.

The following example policies might lead to workload deployment complexities:

- **Policy:** *Secrets [in Key Vault] should have the specified maximum validity period.*

Complication: Foundry can store secrets related to knowledge and tool connections in a connected Key Vault. Those secrets don't have an expiry date set by the service. The baseline architecture doesn't use these types of connections, but you can extend the architecture to support them.

- **Policy:** *AI Search services should use customer-managed keys to encrypt data at rest.*

Complication: This architecture doesn't require customer-managed keys. But you can extend the architecture to support them.

- **Policy:** *Foundry models should not be preview.*

Complication: During development, you might use a preview model that you expect to be generally available by the time that you enable agent capability in your production workload.

Platform teams might apply DINE policies to handle automated deployments into an application landing zone subscription. Preemptively incorporate and test the platform-initiated restrictions and changes into your IaC templates. If the platform team uses Azure policies that conflict with the requirements of the application, you can negotiate a resolution.

Manage changes over time

In this architecture, platform-provided services and operations serve as external dependencies. The platform team continues to apply changes, onboard landing zones, and apply cost controls. The platform team might not prioritize individual workloads. Changes to those dependencies, such as firewall modifications, can affect multiple workloads.

You must communicate with platform teams in an efficient and timely manner to manage all external dependencies. It's important to test changes beforehand so that they don't negatively affect workloads.

Platform changes that affect the workload

In this architecture, the platform team manages the following resources. Changes to these resources can potentially affect the workload's reliability, security, operations, and performance targets. Evaluate these changes before the platform team implements them to determine how they affect the workload.

- **Azure policies:** Changes to Azure policies can affect workload resources and their dependencies. These changes might include direct policy updates or movement of the landing zone into a new management group hierarchy. These changes might go unnoticed until a new deployment occurs, so you must thoroughly test them.

Example: A policy no longer allows the deployment of Azure Cosmos DB instances that require customer-managed key encryption, and your architecture uses Microsoft-managed key encryption.

- **Firewall rules:** Modifications to firewall rules can affect the workload's virtual network or rules that apply broadly across all traffic. These modifications can result in blocked traffic and even silent process failures. These potential problems apply to both the egress firewall and Azure Virtual Network Manager-applied NSG rules.

Example: Blocked traffic to Bing APIs leads to failed agent tool invocations for internet grounding data.

- **Routing in the hub network:** Changes in the transitive nature of routing in the hub can potentially affect the workload functionality if a workload depends on routing to other virtual networks.

Example: A routing change prevents Foundry agents from accessing a vector store that's operated by another team or prevents data science teams from accessing the Foundry portal from their workstations.

- **Azure Bastion host:** Changes to the Azure Bastion host availability or configuration.

Example: A configuration change prevents access to jump boxes and build agent VMs.

Workload changes that affect the platform

The following examples describe workload changes that you should communicate to the platform team. The platform team must validate the platform service's reliability, security, operations, and performance targets against your new changes before they go into effect.

- **Network egress:** Monitor any significant increase in network egress to prevent the workload from becoming a noisy neighbor on network devices. This problem can potentially affect the performance or reliability targets of other workloads. This architecture is mostly self-contained and is unlikely to experience a significant change in outbound traffic patterns.
- **Public access:** Changes to public access for workload components might require extra testing. The platform team might relocate the workload to a different management group.

Example: In this architecture, if you remove the public IP address from Application Gateway and make this application internal only, the workload's exposure to the internet changes. Another example is exposing the browser-based AI portals to the internet, which we don't recommend.

- **Business criticality rating:** Changes to the workload's service-level agreements (SLAs) might require a new collaboration approach between the platform and workload teams.

Example: Your workload might transition from low to high business critically because of increased adoption and success.

- **Data sovereignty:** Changes to requirements around data sovereignty might require the platform team to change their log collection plan, their support of your workload's

Purview integration, or their support of the organization's security information and event management (SIEM) solution.

Example: Your workload might now require that all agent thread data stay within a geographical boundary. If your Foundry is connected to Purview or a SIEM, you'd need to ensure that your users' data doesn't get replicated to a region that violates your regulatory requirements.

Enterprise architecture team

Some organizations have an enterprise architecture team that might influence the design of your workload and its architecture. That team understands the enterprise's [AI adoption](#) strategy and the principles and recommendations in the [AI workloads on Azure](#) design. Work with this team to ensure that this chat workload meets scenario-specific objectives and aligns with organizational strategy and recommendations.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Well-Architected Framework](#).

Reliability

Reliability helps ensure that your application can meet the commitments that you make to your customers. For more information, see [Design review checklist for Reliability](#).

This architecture maintains the [reliability guarantees in the baseline architecture](#). It doesn't introduce new reliability considerations for the core workload components.

Critical dependencies

Treat all functionality that the workload performs in the platform and application landing zone as dependencies. Maintain incident response plans that include contact methods and escalation paths for each dependency. Include these dependencies in the workload's failure mode analysis (FMA).

Consider the following workload dependencies and example scenarios that might occur:

- **Egress firewall:** The centralized egress firewall undergoes changes unrelated to the workload. Multiple workloads share the firewall.

- **DNS resolution:** This architecture uses DNS provided by the platform team for most resources, combined with Azure DNS and linked DNS Private Resolver rulesets for Agent Service. As a result, the workload depends on timely updates to DNS records for private endpoints and availability of DNS services.
- **DINE policies:** DINE policies for Azure private DNS zones, or any other platform-provided dependency, operate on a *best-effort* basis and don't include an SLA. For example, a delay in DNS configuration for this architecture's private endpoints can prevent the chat UI from becoming traffic-ready or block agents from completing Agent Service queries.
- **Management group policies:** Consistent policies among environments support reliability. Ensure that preproduction environments match production environments to provide accurate testing and prevent environment-specific deviations that can block a deployment or scale. For more information, see [Manage application development environments in Azure landing zones](#).

Many of these considerations might exist without Azure landing zones. You need to work with platform teams collaboratively to address these problems and ensure that you meet all requirements. For more information, see [Identify dependencies](#).

Security

Security provides assurances against deliberate attacks and the misuse of your valuable data and systems. For more information, see [Design review checklist for Security](#).

Ingress traffic control

To isolate your workload from other workload spokes within your organization, apply NSGs on your subnets and use the nontransitive nature or controls in the regional hub. Construct comprehensive NSGs that only permit the inbound network requirements of your application and its infrastructure. We recommend that you don't solely rely on the nontransitive nature of the hub network for security.

The platform team implements Azure policies for security. For example, a policy might ensure that Application Gateway has a web application firewall set to *deny mode*, which restricts the number of public IP addresses available to your subscription. In addition to those policies, you should deploy more workload-centric policies that reinforce the ingress security posture.

The following table shows examples of ingress controls in this architecture.

 Expand table

| Source | Purpose | Workload control | Platform control |
|----------------|--|---|---|
| Internet | Application traffic flows | Funnel all workload requests through an NSG, a web application firewall, and routing rules before allowing public traffic to transition to private traffic for the chat UI. | None |
| Internet | Foundry portal access and data plane REST API access | Deny all access through service-level configuration. | None |
| Internet | Data plane access to all services except Application Gateway | Deny all access through NSG rules and service-level configuration. | None |
| Azure Bastion | Jump box and build agent access | Apply an NSG on the jump box subnet that blocks all traffic to remote access ports, unless the source is the platform's designated Azure Bastion subnet. | None |
| Cross-premises | Foundry portal access and data plane REST API access | Deny all access. If you don't use the jump box, allow access only from workstations in authorized subnets for data scientists. | Enforce nontransitive routing or use Azure Firewall in an Azure Virtual WAN secured hub |
| Other spokes | None | Blocked via NSG rules. | Enforce nontransitive routing or use Azure Firewall in a Virtual WAN secured hub |

Egress traffic control

Apply NSG rules that express the required outbound connectivity requirements of your solution and deny everything else. Don't rely only on the hub network controls. As a workload operator, you must stop undesired egress traffic as close to the source as possible.

You own your workload's subnets within the virtual network, but the platform team likely created firewall rules to specifically represent your captured requirements as part of your subscription vending process. Ensure that changes in subnets and resource placement over the lifetime of your architecture remain compatible with your original request. Work with your network team to ensure continuity of least-access egress control.

The following table shows examples of egress controls in this architecture.

 Expand table

| Endpoint | Purpose | Workload control | Platform control |
|-------------------------|--|--|--|
| Public internet sources | Your agent might require internet search to ground an Azure OpenAI request | Apply NSGs on the agent integration subnet | Apply firewall application rules for external knowledge stores and tools |
| Foundry data plane | The chat UI interacts with the chat agent | Allow TCP/443 from the App Service integration subnet to the Foundry private endpoint subnet | None |
| Azure Cosmos DB | To access the memory database from Agent Service | Allow TCP on every port to the Azure Cosmos DB private endpoint subnet | None |

For traffic that leaves the workload's virtual network, this architecture applies controls at the workload level via NSGs and at the platform level via a hub network firewall. The NSGs provide initial, broad network traffic rules. In the platform's hub, the firewall applies more specific rules for added security.

This architecture doesn't require east-west traffic between internal components, such as Agent Service and its dependent AI Search instance, to route through the hub network.

DDoS Protection

Determine who should apply the DDoS Protection plan that covers your solution's public IP addresses. The platform team might use IP address protection plans, or they might use Azure Policy to enforce virtual network protection plans. This architecture requires DDoS Protection coverage because it has a public IP address for ingress from the internet. For more information, see [Recommendations for networking and connectivity](#).

Identity and access management

Unless the platform team's governance automation requires extra controls, this architecture doesn't introduce new authorization requirements because of the platform team's involvement. Azure role-based access control (Azure RBAC) should continue to fulfill the principle of least privilege, which grants limited access only to individuals who need it and only when needed. For more information, see [Recommendations for identity and access management](#).

Certificates and encryption

Your team typically procures the TLS certificate for the public IP address on Application Gateway. Work with the platform team to understand how the certificate procurement and management processes should align with organizational expectations.

All data storage services in this architecture support Microsoft-managed or customer-managed encryption keys. Use customer-managed encryption keys if your workload design or organization requires more control. Azure landing zones themselves don't mandate a specific method.

Microsoft Defender for Cloud

Use the same configuration for Microsoft Defender for Cloud as discussed in the [baseline architecture](#). If your subscription vending process doesn't automatically enable these Defender plans, ensure you take on this responsibility as the workload team.

Microsoft Purview

It's expected that you'll be required to use the native integration of Microsoft Purview when deploying Foundry. During deployment, ensure you enable Microsoft Purview Data Security in the Microsoft Foundry Control Plane. See [Use Microsoft Purview to manage data security & compliance for Microsoft Foundry](#).

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

All [cost optimization strategies in the baseline architecture](#) apply to the workload resources in this architecture.

This architecture greatly benefits from Azure landing zone [platform resources](#). For example, resources such as Azure Firewall and DDoS Protection transition from workload to platform

resources. Even if you use those resources through a chargeback model, the added security and cross-premises connectivity are more cost-effective than self-managing those resources. Take advantage of other centralized offerings from your platform team to extend those benefits to your workload without compromising its service-level objective, recovery time objective, or recovery point objective.

Important

Don't try to optimize costs by consolidating Foundry dependencies as platform resources. These services must remain workload resources.

Operational Excellence

Operational Excellence covers the operations processes that deploy an application and keep it running in production. For more information, see [Design review checklist for Operational Excellence](#).

You remain responsible for all [operational excellence considerations from the baseline architecture](#). These responsibilities include monitoring, GenAIOps, quality assurance, and safe deployment practices.

Correlate data from multiple sinks

The workload's Azure Monitor Logs workspace stores the workload's logs and metrics and its infrastructure components. However, a central Azure Monitor Logs workspace often stores logs and metrics from centralized services, such as Azure Firewall, DNS Private Resolver, and Azure Bastion. Correlating data from multiple sinks can be a complex task.

Correlated data helps support incident response. The triage runbook for this architecture should address this situation and include organizational contact information if the problem extends beyond workload resources. Workload administrators might require assistance from platform administrators to correlate log entries from enterprise networking, security, or other platform services.

Important

For the platform team: When possible, grant Azure RBAC permissions to query and read log sinks for relevant platform resources. Enable firewall logs for network and application rule evaluations and DNS proxy. The application teams can use this information to

troubleshoot tasks. For more information, see [Recommendations for monitoring and threat detection](#).

Build agents

Many services in this architecture use private endpoints. Similar to the baseline architecture, this design might require build agents. Your team deploys the build agents safely and reliably. The platform team isn't involved in this process.

Make sure that the build agent management complies with organizational standards. These standards might include the use of platform-approved operating system images, patching schedules, compliance reporting, and user authentication methods.

Performance Efficiency

Performance Efficiency refers to your workload's ability to scale to meet user demands efficiently. For more information, see [Design review checklist for Performance Efficiency](#).

The [performance efficiency considerations in the baseline architecture](#) also apply to this architecture. Your team retains control over the resources in the application flows, not the platform team. Scale the chat UI host, language models, and other components according to the workload and cost constraints. Depending on the final implementation of your architecture, consider the following factors when you measure your performance against performance targets:

- Egress and cross-premises latency
- SKU limitations from cost containment governance

Deploy this scenario

Deploy a landing zone implementation of this reference architecture.

[Agent Service chat baseline reference implementation](#)

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal authors:

- [Bilal Amjad](#) | Microsoft Cloud Solution Architect

- [Freddy Ayala](#) | Microsoft Cloud Solution Architect
- [Chad Kittel](#) | Principal Software Engineer - Azure Patterns & Practices

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next step

Learn how to collaborate on technical details with the platform team.

[Subscription vending](#)

Related resource

- A Well-Architected Framework perspective on [AI workloads on Azure](#)

Automate document classification in Azure

Azure Functions

Azure AI Foundry

Azure AI Foundry SDK

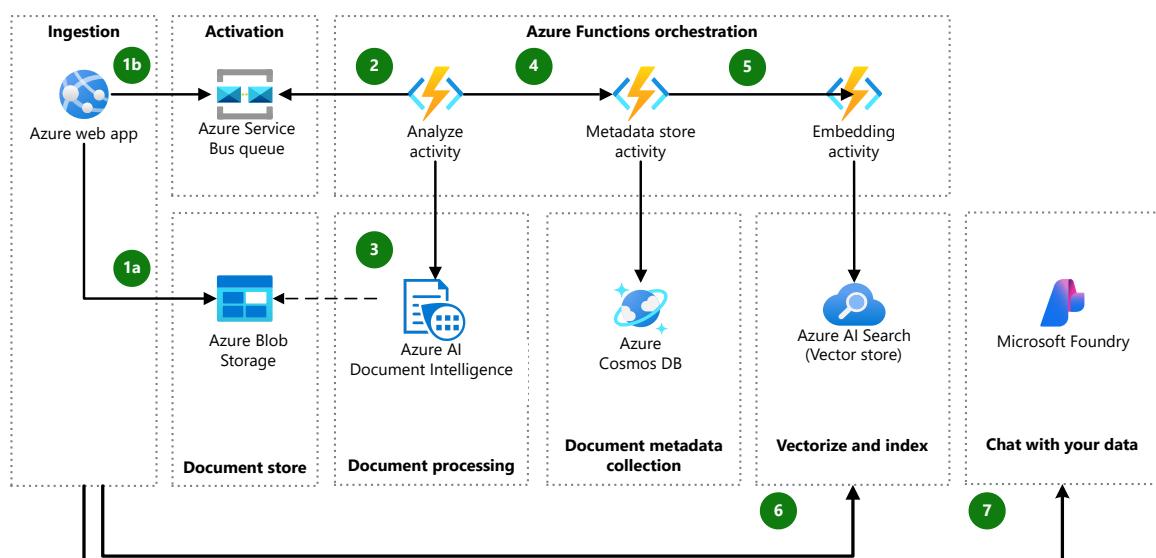
Azure AI services

Azure AI Search

Azure AI Document Intelligence

This article describes an architecture that processes various documents. The architecture uses the durable functions feature of Azure Functions to implement pipelines. The pipelines process documents via Azure AI Document Intelligence for document splitting, named entity recognition (NER), and classification. Retrieval-augmented generation (RAG)-based natural language processing (NLP) uses document content and metadata to find and generate relevant information.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

The following workflow corresponds to the previous diagram:

1. A user uploads a document file to a web app. The file contains multiple embedded documents of various types, like PDF or multiple-page Tag Image File Format (TIFF) files. Azure Blob Storage stores the document file (1a). To initiate pipeline processing, the web app adds a command message to an Azure Service Bus queue (1b).

2. The command message triggers the durable functions orchestration. The message contains metadata that identifies the Blob Storage location of the document file to process. Each durable functions instance processes only one document file.
3. The *analyze* activity function calls the Document Intelligence Analyze Document API, which passes the storage location of the document file to process. The analyze function reads and identifies each document within the document file. This function returns the name, type, page ranges, and content of each embedded document to the orchestration.
4. The *metadata store* activity function saves the document type, location, and page range information for each document in an Azure Cosmos DB store.
5. The *embedding* activity function uses Semantic Kernel to chunk each document and create embeddings for each chunk. The function sends the embeddings and associated content to Azure AI Search and stores them in a vector-enabled index. The function also adds a correlation ID to the search document so that the search results match the corresponding document metadata from Azure Cosmos DB.
6. Semantic Kernel retrieves embeddings from the AI Search vector store for NLP.
7. Users can chat with their data by using NLP. Grounded data retrieved from the vector store powers this conversation. To look up document records in Azure Cosmos DB, users use correlation IDs included in the search result set. The records include links to the original document file in Blob Storage.

Components

- [Durable functions](#) is a feature of [Azure Functions](#) that you can use to write stateful functions in a serverless compute environment. In this architecture, a message in a Service Bus queue triggers a durable functions instance. This instance then initiates and orchestrates the document-processing pipeline.
- [Azure Cosmos DB](#) is a globally distributed, multiple-model database that can scale throughput and storage capacity across any number of geographic regions. Comprehensive service-level agreements (SLAs) guarantee throughput, latency, availability, and consistency. In this architecture, Azure Cosmos DB serves as the metadata store for the document classification information.
- [Azure Storage](#) is a set of scalable and secure cloud services for data, apps, and workloads. It includes [Blob Storage](#), [Azure Files](#), [Azure Table Storage](#), and [Azure Queue Storage](#). In this architecture, Blob Storage stores the document files that the user uploads and that the durable functions pipeline processes.

- [Service Bus](#) is a managed enterprise message broker that has message [queues](#) and publish-subscribe [topics](#). In this architecture, Service Bus triggers durable functions instances.
- [Azure App Service](#) provides a framework to build, deploy, and scale web apps. The Web Apps feature of App Service is an HTTP-based tool that hosts web applications, REST APIs, and mobile back ends. You can use Web Apps to develop in .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. Applications can run and scale in Windows-based and Linux-based environments. In this architecture, users interact with the document-processing system through an App Service-hosted web app.
- [Document Intelligence](#) is a service that extracts insights from your documents, forms, and images. This architecture uses Document Intelligence to analyze the document files and extract the embedded documents along with content and metadata information.
- [AI Search](#) provides a search experience for private, diverse content in web, mobile, and enterprise applications. In this architecture, AI Search [vector storage](#) indexes embeddings of the extracted document content and metadata information so that users can search and retrieve documents by using NLP.
- [Semantic Kernel](#) is a framework that integrates large language models (LLMs) into applications. In this architecture, Semantic Kernel creates embeddings for the document content and metadata information, which are stored in AI Search.
- [Microsoft Foundry](#) is a platform that you use to build, test, and deploy AI solutions and models as a service (MaaS). In this architecture, Foundry deploys an Azure OpenAI model.
 - [Foundry projects](#) are specialized workspaces that you can use to establish connections to data sources, define agents, and invoke deployed models, including Azure OpenAI models. This architecture has a single Foundry project within the Foundry account.
 - [Foundry Models](#) is a platform that deploys flagship models, including OpenAI models, from the Azure AI catalog in a Microsoft-hosted environment. This approach uses MaaS deployment. This architecture deploys models by using the [Global Standard](#) configuration with a fixed quota.

Alternatives

- To facilitate global distribution, this solution stores metadata in Azure Cosmos DB. [Azure SQL Database](#) is another persistent storage option for document metadata and information.

- To trigger durable functions instances, you can use other messaging platforms, including [Azure Event Grid](#).
- Instead of Semantic Kernel, you can use [Azure Machine Learning](#) or [Azure AI services](#) to create embeddings.
- You can use the [Microsoft Agent Framework](#) instead of Semantic Kernel to orchestrate the workflows.
- To provide a natural language interface for users, you can use other language models within Foundry. The platform supports various models from different providers, including Mistral, Meta, Cohere, and Hugging Face.

Scenario details

In this architecture, the pipelines identify the documents in a document file, classify them by type, and store information to use in subsequent processing.

Many companies need to manage and process documents that they scan in bulk and that contain several different document types, like PDFs or multiple-page TIFF images. These documents might originate from outside the organization, and the receiving company doesn't control the format.

Because of these constraints, organizations must build their own document-parsing solutions that can include custom technology and manual processes. For example, someone might manually separate individual document types and add classification qualifiers for each document type.

Many of these custom solutions are based on the state machine workflow pattern. The solutions use database systems to persist workflow state and use polling services that check for the states that they need to process. Maintaining and enhancing these solutions can increase complexity and effort.

Organizations need reliable, scalable, and resilient solutions to process and manage document identification and classification for their organization's document types. This solution can process millions of documents each day with full observability into the success or failure of the processing pipeline.

NLP lets users interact with the system in a conversational manner. Users can ask questions about the documents and receive answers based on the content of the documents.

Potential use cases

- **Generate report titles.** Many government agencies and municipalities manage paper records that don't have a digital form. An effective automated solution can generate a file that contains all the documents that you need to satisfy a document request.
- **Manage maintenance records.** Scan and send paper records, like aircraft, locomotive, and machinery maintenance records, to outside organizations.
- **Process permits.** City and county permitting departments maintain paper documents that they generate for permit inspection reporting. You can take a picture of several inspection documents and automatically identify, classify, and search across these records.
- **Analyze planograms.** Retail and consumer goods companies manage inventory and compliance through store shelf planogram analysis. You can take a picture of a store shelf and extract label information from different products to automatically identify, classify, and quantify the product information.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Well-Architected Framework](#).

Reliability

Reliability helps ensure that your application can meet the commitments that you make to your customers. For more information, see [Design review checklist for Reliability](#).

To ensure reliability and high availability when you invoke models from Foundry projects that use OpenAI models hosted in Azure, consider using a generative API gateway like [Azure API Management](#). This approach manages requests across multiple model deployments or Foundry endpoints. The Azure back-end gateway supports round-robin, weighted, and priority-based routing across deployments and provides full control of traffic distribution. This approach lets your Foundry project implement resilient failover strategies and intelligent load distribution tuned to your performance, regional availability, or cost requirements.

For learning and early proof-of-concept work, use a [Global Standard](#) deployment. Global Standard is pay-as-you-go, provides the highest default quota, and uses the Azure global infrastructure to route each request to the most available region. This approach reduces the chance of encountering regional quota or capacity constraints while you experiment and aligns with the Microsoft guidance to use Global Standard as the default starting point.

For production workloads, choose the [deployment type](#) based on the following criteria:

- **Data-processing location:**
 - Use *Global Standard* or *Global Provisioned* when you want the highest availability and inferencing can occur in any Microsoft Foundry region, while data at rest remains in your selected geography.
 - Use *Data Zone Standard* or *Data Zone Provisioned* when you must keep inferencing within a Microsoft-defined data zone, for example US-only or EU-only, to meet data residency requirements.
- **Throughput and cost model:**
 - Use *Standard deployment types*, like *Global Standard*, *Data Zone Standard*, and *Regional Standard* for low-to-medium, bursty, or exploratory workloads. These types use a pay-as-you-go model with no reserved capacity. Choose these types in early stages before you understand your traffic patterns.
 - Use *Provisioned deployment types*, like *Global Provisioned*, *Data Zone Provisioned*, and *Regional Provisioned* for predictable, higher-volume workloads that need reserved throughput, consistent latency, and the option to use reservations for cost optimization.

Most teams begin with **Global Standard** for development, or use **Data Zone Standard** when data residency is important. After they determine their steady-state throughput and latency requirements, they move critical paths to **Provisioned** SKUs.

For more information about reliability in solution components, see [SLA information for Azure online services](#).

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

The most significant costs for this architecture include the following components:

- Model inference usage via Microsoft Foundry, which includes OpenAI or other models
- Document ingestion and processing via Document Intelligence
- Indexing and search consumption via AI Search

To optimize costs, consider the following recommendations:

- **Use provisioned throughput units (PTUs) or reservations for Microsoft Foundry deployments instead of pay-per-token usage when the workload is predictable.**

- For more information, see the following resources:
 - [Provisioned throughput overview](#)
 - [Save costs with Microsoft Foundry reservations](#)
 - [Plan and manage Microsoft Foundry costs](#)
- Plan for [regional deployments and operational scale-up scheduling](#) in AI Search.
- Use [commitment tier pricing](#) for Document Intelligence to manage [predictable costs](#).
- Use [reserved capacity and life cycle policies](#) to [rightsize storage accounts](#).
- Use the [pay-as-you-go strategy](#) for your architecture and [scale out](#) as needed instead of investing in large-scale resources at the start. As your solution matures, you can use [App Service reservations](#) to help reduce costs where applicable.
- Consider opportunity costs in your architecture and balance a first-mover advantage strategy with a fast-follow strategy. To estimate the initial cost and operational costs, use the [Azure pricing calculator](#).
- Establish [budgets and controls](#) that set cost limits for your solution. To set up forecasting and actual cost alerts, use [budget alerting](#).

Performance Efficiency

Performance Efficiency refers to your workload's ability to scale to meet user demands efficiently. For more information, see [Design review checklist for Performance Efficiency](#).

This solution can expose performance bottlenecks when you process high volumes of data. To ensure proper performance efficiency for your solution, understand and plan for [Azure Functions scaling options](#), [AI services autoscaling](#), and [Azure Cosmos DB partitioning](#).

- Apply [scalable compute and orchestration](#) by using durable functions, which is part of Azure Functions, for the document-processing pipeline and tune its scaling behavior. For more information, see [Performance and scale in durable functions](#).
- Choose the appropriate deployment model in Microsoft Foundry for inference workloads. Use serverless APIs for variable workloads and provisioned throughput models when you expect heavy, consistent traffic. For more information, see [Provisioned throughput for Foundry Models](#) and [Performance and latency optimization for Azure OpenAI and Foundry Models](#).
- Optimize [indexing and retrieval performance](#) by configuring appropriate partitioning, replicas, and schema for AI Search. For more information, see [AI Search performance tips](#).

- Establish performance baselines and feedback loops. Define realistic latency and throughput targets early, monitor actual system performance continuously, and refine architecture and operational configurations as usage patterns evolve. For more information, see [Performance Efficiency design principles](#).

Apply these practices to help ensure that your document classification solution remains responsive and cost effective as the solution scales.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal author:

- [Kevin Kraus](#) | Principal Solution Engineer

Other contributor:

- [Brian Swiger](#) | Principal Solution Engineer

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

The following articles provide an introduction to relevant technologies:

- [What is Blob Storage?](#)
- [What is Service Bus?](#)
- [Get started with App Service](#)
- [Introduction to Azure Cosmos DB](#)

For product documentation, see the following resources:

- [Azure documentation for all products](#)
- [Durable functions documentation](#)
- [Microsoft Foundry documentation](#)
- [Document Intelligence documentation](#)
- [AI Search documentation](#)
- [Semantic Kernel documentation](#)

Related resources

- [Custom document processing models on Azure](#)

- Image classification on Azure

Automate PDF forms processing

Azure AI Document Intelligence

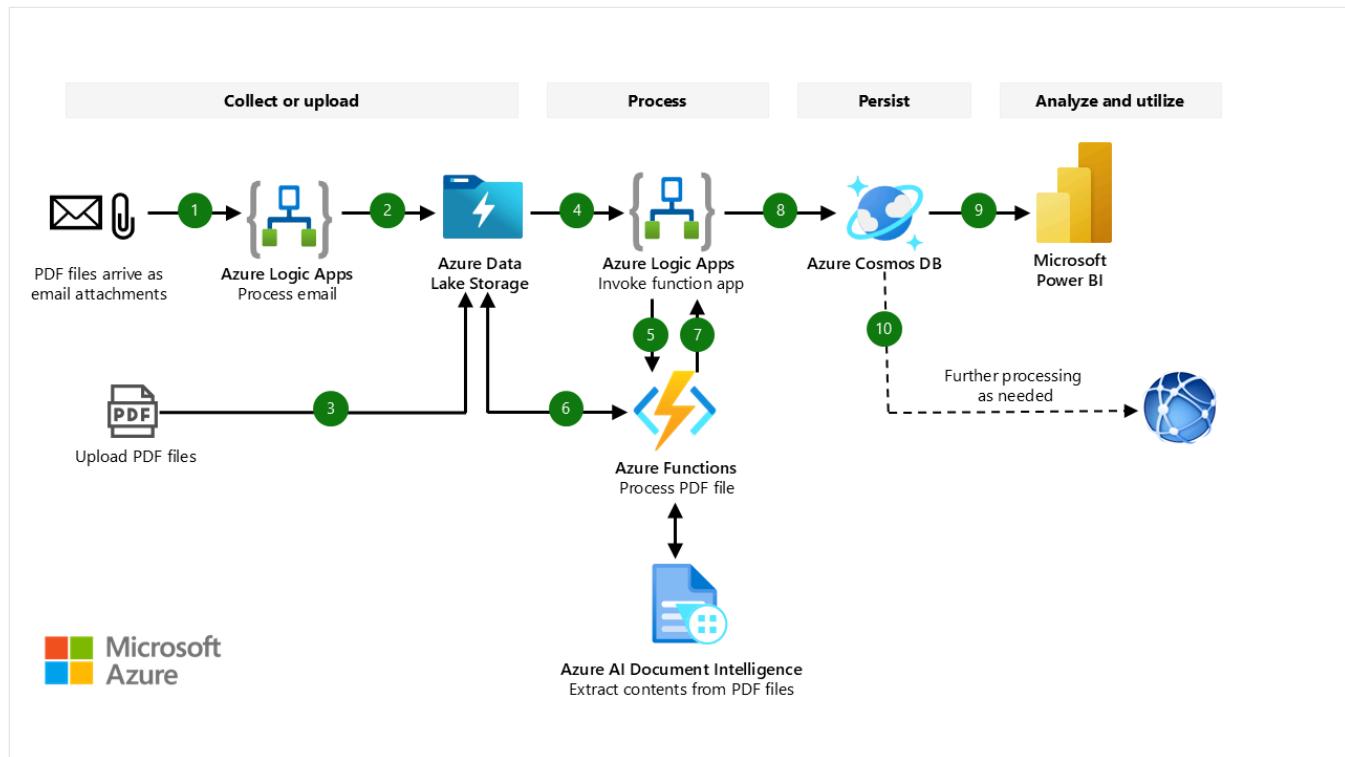
Azure AI services

Azure Logic Apps

Azure Functions

This article describes an Azure architecture that you can use to replace costly and inflexible forms processing methods with cost-effective and flexible automated PDF processing.

Architecture



Download a [PowerPoint file](#) of this architecture.

Workflow

1. A designated Outlook email account receives PDF files as attachments. The arrival of an email triggers a logic app to process the email. The logic app is built by using the capabilities of Azure Logic Apps.
2. The logic app uploads the PDF files to a container in Azure Data Lake Storage.
3. You can also manually or programmatically upload PDF files to the same PDF container.
4. The arrival of a PDF file in the PDF container triggers another logic app to process the PDF forms that are in the PDF file.
5. The logic app sends the location of the PDF file to a function app for processing. The function app is built by using the capabilities of Azure Functions.
6. The function app receives the location of the file and takes these actions:
 - a. It splits the file into single pages if the file has multiple pages. Each page contains one independent form. Split files are saved to a second container in Data Lake Storage.

- b. It uses HTTPS POST, an Azure REST API, to send the location of the single-page PDF file to AI Document Intelligence for processing. When Azure AI Document Intelligence completes its processing, it sends a response back to the function app, which places the information into a data structure.
 - c. It creates a JSON data file that contains the response data and stores the file to a third container in Data Lake Storage.
7. The forms processing logic app receives the processed response data.
 8. The forms processing logic app sends the processed data to Azure Cosmos DB, which saves the data in a database and in collections.
 9. Power BI obtains the data from Azure Cosmos DB and provides insights and dashboards.
 10. You can implement further processing as needed on the data that's in Azure Cosmos DB.

Components

- [Azure AI Document Intelligence](#), a cloud-based service that enables you to build intelligent document processing solutions. It applies advanced machine learning to extract text, key-value pairs, tables, and structures from documents automatically and accurately. In this architecture, it is the intelligent document processing service utilized to extract information from PDF documents.
- [Azure Logic Apps](#) is a serverless cloud service for creating and running automated workflows that integrate apps, data, services, and systems. In this architecture, it is used to as an orchestrator to coordinate the user input, document storage, document processing, storage of the results, and analysis of the processed documents.
- [Azure Functions](#) is a serverless solution that makes it possible for you to write less code, maintain less infrastructure, and save on costs. In this architecture, it is the backend services to configure input to utilize [Azure AI Document Intelligence](#), and store the output.
- [Azure Data Lake Storage](#) is the foundation for building enterprise data lakes on Azure. In this architecture, it is used to store the raw PDF documents, machine learning results, and processed output.
- [Azure Cosmos DB](#) is a fully managed NoSQL and relational database for modern app development. In this architecture, it is used to store the extracted insights from each PDF document. The information is utilized by [Power BI](#) to produce insights.
- [Power BI](#) is a collection of software services, apps, and connectors that work together so that you can turn your unrelated sources of data into coherent, visually immersive, and interactive insights. In this architecture, it is used to analyze the document processing results.

Alternatives

- You can use [Microsoft Fabric](#) instead of [Power BI](#) to ingest the processed output to a Lakehouse and then perform further analysis and processing of the output data.

Scenario details

Forms processing is often a critical business function. Many companies still rely on manual processes that are costly, time consuming, and prone to error. Replacing manual processes reduces cost and risk and makes a company more agile.

This article describes an architecture that you can use to replace manual PDF forms processing or costly legacy systems that automate PDF forms processing. Azure AI Document Intelligence processes the PDF forms, Logic Apps provides the workflow, and Functions provides data processing capabilities.

For deployment information, see [Deploy this scenario](#) in this article.

Potential use cases

The solution that's described in this article can process many types of forms, including:

- Invoices and payment records
- Purchase orders
- Safety, incident, and compliance records
- Health screening forms
- Customer feedback forms
- Employee records
- Academic and research papers
- Documents with handwritten notes
- Custom documents from your domain

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

Reliability

Reliability ensures that your application can meet the commitments that you make to your customers. For more information, see [Design review checklist for Reliability](#).

This architecture is intended as a starter architecture that you can quickly deploy and prototype to provide a business solution. If your prototype is a success, you can then extend and enhance the architecture, if necessary, to meet additional requirements.

This architecture utilizes scalable and resilient Azure infrastructure and technologies. For example, Azure Cosmos DB has built-in redundancy and global coverage that you can configure to meet your needs.

For the availability guarantees of the Azure services that this solution uses, see [Service-level agreements \(SLAs\) for Online Services](#).

Security

Security provides assurances against deliberate attacks and the abuse of your valuable data and systems. For more information, see [Design review checklist for Security](#).

The Outlook email account that's used in this architecture is a dedicated email account that receives PDF forms as attachments. It's good practice to limit the senders to trusted parties only and to prevent malicious actors from spamming the email account.

The implementation of this architecture that's described in [Deploy this scenario](#) takes the following measures to increase security:

- The PowerShell and Bicep deployment scripts use Azure Key Vault to store sensitive information so that it isn't displayed on terminal screens or stored in deployment logs.
- Managed identities provide an automatically managed identity in Microsoft Entra ID for applications to use when they connect to resources that support Microsoft Entra authentication. The function app uses managed identities so that the code doesn't depend on individual principals and doesn't contain sensitive identity information.

Cost Optimization

Cost Optimization is about looking at ways to reduce unnecessary expenses and to improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

Here are some guidelines for optimizing costs:

- Use the pay-as-you-go strategy for your architecture, and [scale out](#) as needed rather than investing in large-scale resources at the start.
- The implementation of the architecture that's described in [Deploy this scenario](#) deploys a starting solution that's suitable for proof of concept. The deployment scripts create a

working architecture with minimal resource requirements. For example, the deployment scripts create a smallest serverless Linux host to run the function app.

Performance Efficiency

Performance Efficiency is the ability of your workload to scale in an efficient manner to meet the demands that are placed on it by users. For more information, see [Design review checklist for Performance Efficiency](#).

This architecture uses services that have built-in scaling capabilities that you can use to improve performance efficiency. Here are some examples:

- You can host both Azure Logic Apps and Azure Functions in a serverless infrastructure. For more information, see [Azure serverless overview: Create cloud-based apps and solutions with Azure Logic Apps and Azure Functions](#).
- You can configure Azure Cosmos DB to automatically scale its throughput. For more information, see [Provision autoscale throughput on a database or container in Azure Cosmos DB - API for NoSQL](#).

Deploy this scenario

You can deploy a rudimentary version of this architecture and use it as a starting point for deploying your own solution. The repository includes code, deployment scripts, and a deployment guide.

The sample receives the PDF forms, extracts the data fields, and saves the data in Azure Cosmos DB. Power BI visualizes the data. The design uses a modular, metadata-driven methodology. No form fields are hard-coded. It can process any PDF forms.

You can use the repository as is, without code modification, to process and visualize any single-page PDF forms such as safety forms, invoices, incident records, and many others. To use it, you only need to collect sample PDF forms, train a new model to learn the layout of the forms, and plug the model into the solution. You also need to redesign the Power BI report for your datasets so that it provides the insights that you want.

The implementation uses [Azure AI Document Intelligence Studio](#) to create custom models. The sample uses the field names that are saved in the machine learning model as a reference to process other forms. Only five sample forms are needed to create a custom-built machine learning model. You can merge as many as 100 custom-built models to create a composite machine learning model that can process a variety of forms.

Deployment repository

The code for this sample is in [Azure PDF Form Processing Automation Solution](#) GitHub repository. Follow the deployment guide in the repository.

Deployment considerations

To process a new type of PDF form, you use sample PDF files to create a new machine learning model. When the model is ready, you plug the model ID into the solution.

This container name is configurable in the deployment scripts that you get from the GitHub repository.

The architecture doesn't address any high availability (HA) or disaster recovery (DR) requirements. If you want to extend and enhance the current architecture for production deployment, consider the following recommendations and best practices:

- Design the HA/DR architecture based on your requirements and use the built-in redundancy capabilities where applicable.
- Update the Bicep deployment code to create a computing environment that can handle your processing volumes.
- Update the Bicep deployment code to create more instances of the architecture components to satisfy your HA/DR requirements.
- Follow the guidelines in [Azure Storage redundancy](#) when you design and provision storage.
- Follow the guidelines in [Business continuity and disaster recovery](#) when you design and provision the logic apps.
- Follow the guidelines in [Reliability in Azure Functions](#) when you design and provision the function app.
- Follow the guidelines in [Achieve high availability with Azure Cosmos DB](#) when you design and provision a database that was created by using Azure Cosmos DB.
- If you consider putting this system into production to process large volumes of PDF forms, you can modify the deployment scripts to create a Linux host that has more resources. To do so, modify the code inside [deploy-functionsapp.bicep](#)

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Gail Zhou](#) | Principal Software Engineer

Other contributors:

- [Said Nikjou](#) | Sr. Cloud Solution Architect
- [Nalini Chandhi](#) | Principal Software Engineering Manager
- [Steve DeMarco](#) | Principal Technical Specialist
- [Travis Hilbert](#) | Software Engineer II
- [DB Lee](#) | Principal Software Engineer
- [Malory Rose](#) | Sr. Software Engineer
- [Oscar Shimabukuro](#) | Sr. Cloud Solution Architect
- [Echo Wang](#) | Solution Architect

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Video: Azure PDF Form Processing Automation](#).
- [Azure PDF Form Processing Automation Solution](#) GitHub repository
- [Azure Invoice Process Automation Solution](#) GitHub repository
- [Tutorial: Create workflows that process emails by using Azure Logic Apps, Azure Functions, and Azure Storage](#)

Related resources

- [Custom document processing models on Azure](#)
- [Index file content and metadata by using Azure AI Search](#)
- [Automate document identification, classification, and search by using Durable Functions](#)

Build and deploy custom document processing models on Azure

Azure AI Document Intelligence

Azure AI services

Azure Logic Apps

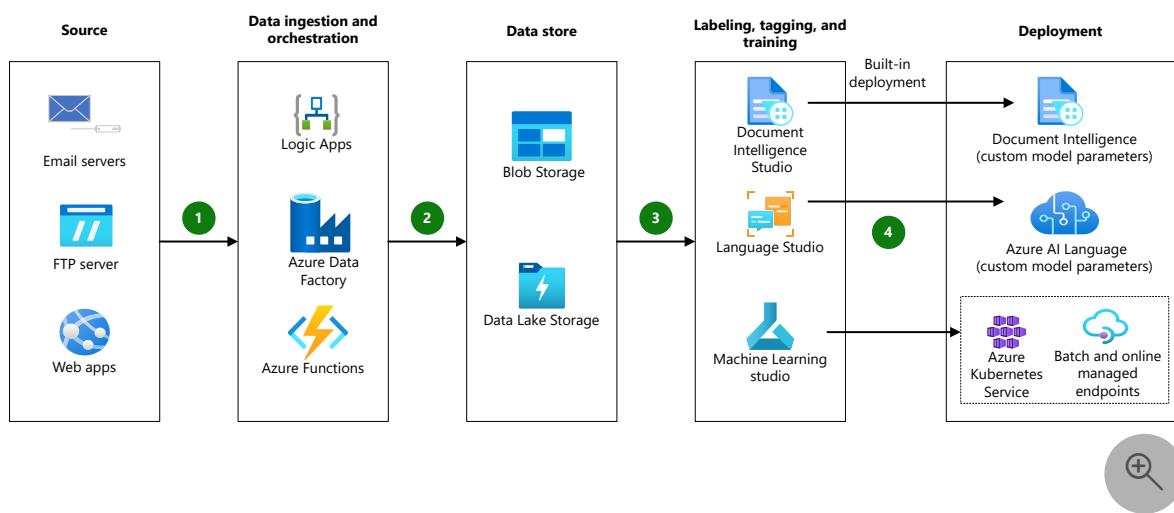
Azure Machine Learning Studio

Azure AI Foundry

Azure Storage

This article describes Azure solutions for building, training, deploying, and using custom document processing models. These Azure services also provide user interface (UI) capabilities for labeling or tagging text during processing.

Architecture



Dataflow

The following dataflow corresponds to the previous diagram:

1. Orchestrators like Azure Logic Apps, Azure Data Factory, or Azure Functions ingest messages and attachments from email servers and files from file transfer protocol servers or web applications.
 - Azure Functions and Azure Logic Apps enable serverless workloads. The service that you choose depends on your preference for service capabilities like development, connectors, management, and operational context. For more information, see [Compare Azure Functions and Azure Logic Apps](#).
 - Consider using Azure Data Factory to move data in bulk.

2. The orchestrators send ingested data to Azure Blob Storage or Azure Data Lake Storage. They organize the data within these stores based on characteristics like file extensions or customer details.
3. You can use the following Azure services, either independently or in combination, for training documents and building custom models to address various use cases.
 - [Document Intelligence Studio](#): If the document requires you to extract key-value pairs or create a custom table from an image or PDF, use Document Intelligence Studio to tag the data and train the custom model. If there's a requirement to identify the type of document, called *document classification*, before you invoke the correct extraction model, use Document Intelligent Studio to label the documents and build the models.
 - [Language Studio](#): For document classification based on content, or for domain-specific entity extraction, you can train a custom text classification or named entity recognition (NER) model in Language Studio.
 - [Azure Machine Learning studio](#): For labeling data for text classification or entity extraction to use with open-source frameworks like PyTorch or TensorFlow, use [Machine Learning studio](#), the [Python SDK](#), [Azure CLI](#), or the [REST API](#). Machine Learning studio provides a [model catalog](#) of foundation models. These foundation models have fine-tuning capabilities for various tasks like text classification, question answering, and summarization. To fine-tune foundation models, use the [Machine Learning studio UI](#) or [code](#).
 - [Azure OpenAI Service](#): To [fine-tune Azure OpenAI models](#) on your own data or domain for various tasks like text summarization and question answering, use [Azure AI Foundry portal](#), [Python SDK](#), or [REST API](#).

4. To deploy the custom models and use them for inferencing:

- Azure AI Document Intelligence has built-in model deployment. Inferencing with the custom models is done by using [SDKs](#) or [document models REST API](#). The `modelId`, or *model name*, specified during model creation is included in the request URL for document analysis. Document Intelligence doesn't require any further deployment steps.
- Language Studio provides an option to deploy custom language models. Get the REST endpoint [prediction URL](#) by selecting the model for deployment. You can inference models by using either the REST endpoint or the Azure SDK client libraries.
- Machine Learning deploys custom models to online or batch [Machine Learning managed endpoints](#). You can also use the Machine Learning SDK to [deploy to Azure](#).

[Kubernetes Service \(AKS\)](#) as a web service. Fine-tuned foundation models can be deployed from the model catalog via managed compute or a [serverless API](#). Models deployed through managed compute can be inferenced by using managed endpoints, which include online endpoints for real-time inferencing and batch endpoints for batch inferencing.

- Azure AI Foundry provides multiple options to [deploy fine-tuned Azure OpenAI models](#). You can deploy these models by using the Python SDK or REST API. You can also deploy fine-tuned foundation models from providers like Meta or Llama as [serverless APIs](#) or by using [managed compute](#).

Components

- [Azure Logic Apps](#) is part of [Azure Integration Services](#). Logic Apps creates automated workflows that integrate apps, data, services, and systems. In this architecture, Logic Apps orchestrates the ingestion of documents and data from various sources and triggers downstream processes for document processing. You can use [managed connectors](#) for services like Azure Storage and Microsoft 365 to trigger workflows when a file arrives in the storage account or an email is received.
- [Azure Data Factory](#) is a managed data integration service for orchestrating and automating data movement and transformation. In this architecture, Azure Data Factory adds [transformation activities](#) like invoking a REST endpoint or running a notebook on the ingested data to the pipeline.
- [Azure Functions](#) is a serverless compute service that can host event-driven workloads that have short-lived processes. In this architecture, Functions enables workloads to process incoming documents and trigger model processing pipelines.
- [Blob Storage](#) is an object storage solution for storing unstructured data. Blob Storage supports libraries for multiple languages, such as .NET, Node.js, and Python. Applications can access files on Blob Storage via HTTP or HTTPS. Blob Storage has [hot, cool, and archive access tiers](#) to support cost optimization for storing large amounts of data. In this architecture, this account is the solution for raw files that use a hot tier.
- [Data Lake Storage](#) is a scalable, cloud-based repository for storing and organizing large volumes of unstructured data. In this architecture, Data Lake Storage organizes and maintains large volumes of ingested data to support analytics, labeling, and machine learning workflows.
- [Document Intelligence](#) is a component of [Azure AI services](#). In this architecture, it provides built-in document analysis capabilities for extracting printed and handwritten text, tables, and key-value pairs. Document Intelligence has prebuilt models for extracting

data from invoices, documents, receipts, ID cards, and business cards. Document Intelligence also has a [custom template](#) form model and a [custom neural document](#) model that you can use to train and deploy custom models.

- [Document Intelligence Studio](#) provides an interface to explore Document Intelligence features and models. You can use the interface to label data and build custom document processing models.
- [Azure AI Language](#) consolidates the Azure natural language processing (NLP) services. It provides [prebuilt and customizable options](#) and language understanding capabilities. Use it to classify documents, recognize named entities, and complete other NLP tasks.
- [Language Studio](#) is a web-based UI in Language that you can use to build, train, manage, and deploy language models. In this architecture, it supports tagging, training, and deploying custom language models for tasks like classification and entity extraction within the document processing pipeline. [Autolabeling](#) supports custom text classification and can automatically label documents into different classes or categories. The studio also provides options to view [model performance](#), including F1 score, precision, and recall.
- [Azure Machine Learning](#) is a managed machine learning platform for model development and deployment at scale. In this architecture, it labels data, trains custom models (including with open-source frameworks), and deploys the models for inference tasks.
 - Machine Learning studio provides data labeling options for [images](#) and [text](#). It supports model training workflows within this architecture.
 - [Export labeled data as COCO](#) or Machine Learning datasets. You can use these datasets to train and deploy models in Machine Learning notebooks.
- [Azure OpenAI](#) provides powerful [language models](#) and [multimodal models](#) as REST APIs that you can use to perform various tasks. In this architecture, Azure OpenAI models perform advanced language tasks such as [fine-tuning models](#) to improve the model performance on data that's missing or underrepresented when the base model is originally trained. You can also use foundation models from multiple providers to perform these tasks.

Alternatives

You can add more workflows to this scenario based on specific use cases.

- If the document is an image or PDF, you can extract the data by using Azure [optical character recognition](#), the [Document Intelligence Read API](#), or open-source libraries.

- You can use the prebuilt model in Language for [document and conversation summarization](#).
- Use preprocessing code to perform text processing steps. These steps include cleaning, stop words removal, lemmatization, stemming, and text summarization on extracted data according to document processing requirements. You can expose the code as REST APIs for automation. Manually complete or automate these steps by integrating with the [Azure Logic Apps](#) or [Azure Functions](#) ingestion process.
- You can use [Azure AI Foundry portal](#) to [fine-tune](#) and deploy foundation models, and build generative AI applications.

Azure AI Foundry provides two compute options for models as a platform (MaaP) hosting, [serverless compute and managed compute](#). [Specific models and regions](#) support deployment through serverless API, which provides models as a service (MaaS).

Machine Learning and Azure AI Foundry share capabilities, so [evaluate both platforms](#) and choose the best one for your scenario.

- You can use [Azure AI Content Understanding](#) to create a [custom analyzer](#) by defining a field schema for extracting structured data from the document.

Scenario details

Document processing covers a wide range of tasks. It can be difficult to meet all your document processing needs by using the prebuilt models available in Language and Document Intelligence. You might need to build custom models to automate document processing for different applications and domains.

Major challenges in model customization include:

- Labeling or tagging text data with relevant key-value pair entities to classify text for extraction.
- Managing training infrastructure, such as compute and storage, and their integrations.
- Deploying models at scale for applications to consume.

Potential use cases

The following use cases can take advantage of custom models for document processing:

- Build custom NER and text classification models based on open-source frameworks.

- Extract custom key values from documents for various industry verticals like insurance and healthcare.
- Tag and extract specific domain-dependent entities beyond the [prebuilt NER models](#) for domains like security or finance.
- Create custom tables from documents.
- Extract signatures.
- Label and classify emails or other documents based on content.
- Summarize documents or create custom question-and-answer models based on your data.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Well-Architected Framework](#).

For this example workload, implementing each pillar depends on optimally configuring and using each component Azure service.

Reliability

Reliability helps ensure that your application can meet the commitments that you make to your customers. For more information, see [Design review checklist for Reliability](#).

Availability

- For more information about the service-level agreements for each architecture component, see [Licensing documents](#).
- For more information about configuration options to design high-availability applications with Storage accounts, see [Use geo-redundancy to design highly available applications](#).

Resiliency

- Address failure modes of individual services like Functions and Storage to help ensure resiliency of the compute services and data stores in this scenario. For more information, see [Reliability guides by service](#).

- Back up and recover your Document Intelligence models.
- Back up and recover your custom [text classification models](#) and [NER models](#) in Language.
- Machine Learning depends on constituent services like Blob Storage, compute services, and AKS. To provide reliability for Machine Learning, configure each of these services to be reliable. For more information on designing for recovery, see [Failover for business continuity and disaster recovery \(BCDR\)](#).
- For Azure OpenAI, help ensure continuous availability by provisioning two or more Azure OpenAI resources in different regions. This approach allows failover to another region if there's a problem. For more information, see [BCDR with Azure OpenAI](#).

Security

Security provides assurances against deliberate attacks and the misuse of your valuable data and systems. For more information, see [Design review checklist for Security](#).

Implement data protection, identity and access management, and network security recommendations for [Blob Storage](#), [AI services](#) for Document Intelligence and Language Studio, [Machine Learning](#), and [Azure OpenAI](#).

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

The total cost of implementing this solution depends on the pricing of the services that you choose.

The major costs for this solution include:

- The compute cost to train and deploy Machine Learning models.

To help optimize costs, choose the right node type, cluster size, and number of nodes. Machine Learning provides options for training, such as setting the minimum number of compute cluster nodes to zero and defining the idle time before scaling down. For more information, see [Manage and optimize Machine Learning costs](#).
- Data orchestration duration and activities. For Azure Data Factory, the charges for copy activities on the Azure integration runtime are based on the number of data integration units used and the time taken to perform the activities. Added orchestration activity runs are also charged, based on their number.

Azure Logic Apps pricing plans depend on the resources that you create and use. The following articles can help you choose the right plan for specific use cases:

- [Costs that typically accrue with Azure Logic Apps](#)
- [Single-tenant versus multitenant environment for Azure Logic Apps](#)
- [Usage metering, billing, and pricing models for Azure Logic Apps](#)

For more information about pricing for specific components, see the following resources:

- [Azure AI Document Intelligence pricing ↗](#)
- [Functions pricing ↗](#)
- [Azure Logic Apps pricing ↗](#)
- [Azure Data Factory pricing ↗](#)
- [Blob Storage pricing ↗](#)
- [Language pricing ↗](#)
- [Machine Learning pricing ↗](#)
- [Azure OpenAI pricing ↗](#)

Use the [Azure pricing calculator ↗](#) to add the component options that you choose and estimate the overall cost of the solution.

Performance Efficiency

Performance Efficiency refers to your workload's ability to scale to meet user demands efficiently. For more information, see [Design review checklist for Performance Efficiency](#).

Scalability

- To scale Functions automatically or manually, [choose the right hosting plan](#).
- By default, Document Intelligence supports 15 concurrent requests per second. To increase this quota, [create an Azure support ticket](#).
- For Machine Learning custom models hosted as web services on AKS, the `azureml-fe` front-end component automatically scales as needed. This component also routes incoming inference requests to deployed services.
- For deployments as managed endpoints, support autoscaling by integrating with the [Azure Monitor autoscale feature](#). For more information, see [Endpoints for inference in production](#).

- The API service limits on [custom NER](#) and [custom text classification](#) for inferencing are 20 GET or POST requests per minute.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal authors:

- Dixit Arora | Senior Engineer
- [Jyotsna Ravi](#) ↗ | Principal Engineer

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Get started with custom projects in Document Intelligence Studio](#)
- [Use Document Intelligence models](#)
- [What is Azure AI Language?](#)
- [What is optical character recognition?](#)
- [How to configure Functions with a virtual network](#)

Related resources

- [Extract text from objects by using Power Automate and AI Builder](#)
- [Suggest content tags with NLP by using deep learning](#)

Create an Azure AI Search index based on file content and metadata

Azure AI Search

Azure Blob Storage

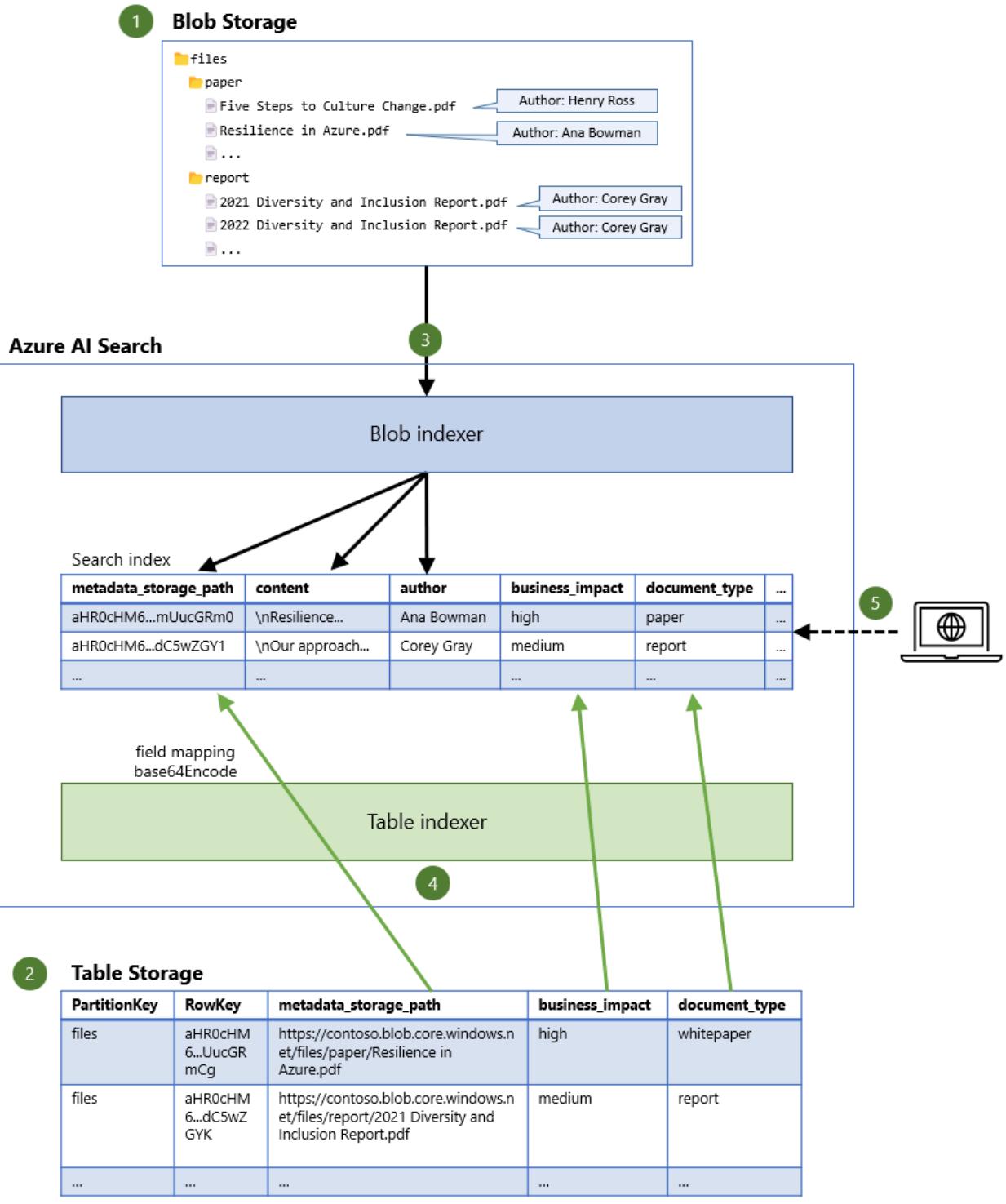
Azure Table Storage

This article demonstrates how to create a search service that enables users to search for documents based on document content in addition to any metadata that's associated with the document.

You can implement this service by using [multiple indexers](#) in Azure AI Search.

This article uses an example workload to demonstrate how to create a single [search index](#) that's based on documents in [Azure Blob Storage](#). The file metadata is stored in [Azure Table Storage](#).

Architecture



Download a [PowerPoint file](#) of this architecture.

Dataflow

1. Documents are stored in Blob Storage, possibly together with a limited amount of metadata (for example, the document's author).
2. Additional metadata is stored in Table Storage, which can store significantly more information for each document.

3. An indexer reads the contents of each file, together with any blob metadata, and stores the data in the search index.
4. Another indexer reads the additional metadata from the table and stores it in the same search index.
5. A search query is sent to the search service. The query returns matching documents, based on both document content and document metadata.

Components

- [Blob Storage](#) provides cost-effective cloud storage for file data, including data in formats like PDF, HTML, and CSV, and in Microsoft 365 documents. In this architecture, Blob Storage is the primary storage location for documents that the solution indexes and searches.
- [Table Storage](#) provides storage for nonrelational structured data. In this architecture, it's used to store the metadata for each document.
- [Azure AI Search](#) is a managed search service that provides infrastructure, APIs, and tools for building a search experience. In this architecture, AI Search indexes document content from Blob Storage and metadata from Table Storage so that users can search across both data sources.

Alternatives

This scenario uses [indexers in Azure AI Search](#) to automatically discover new content in supported data sources, like blob and table storage, and then add it to the search index. Alternatively, you can use the APIs provided by Azure AI Search to [push data to the search index](#). If you do, however, you need to write code to push the data into the search index and also to parse and extract text from the binary documents that you want to search. The [Blob Storage indexer supports many document formats](#), which significantly simplifies the text extraction and indexing process.

Also, if you use indexers, you can optionally [enrich the data as part of an indexing pipeline](#). For example, you can use Azure AI services to perform [optical character recognition \(OCR\)](#) or [visual analysis](#) of the images in documents, [detect the language](#) of documents, or [translate](#) documents. You can also define your own [custom skills](#) to enrich the data in ways that are relevant to your business scenario.

This architecture uses blob and table storage because they're cost-effective and efficient. This design also enables combined storage of the documents and metadata in a single storage account. Alternative supported data sources for the documents themselves include [Azure Data](#)

Lake Storage and Azure Files. Document metadata can be stored in any other supported data source that holds structured data, like Azure SQL Database and Azure Cosmos DB.

Scenario details

Searching file content

This solution enables users to search for documents based on both file content and additional metadata that's stored separately for each document. In addition to searching the text content of a document, a user might want to search for the document's author, the document type (like *paper* or *report*), or its business impact (*high*, *medium*, or *low*).

Azure AI Search is a fully managed search service that can create [search indexes](#) that contain the information you want to allow users to search for.

The documents that are searched in this scenario are stored in [Blob Storage](#). Because of this, you can use the built-in [Blob Storage indexer](#) in Azure AI Search to automatically extract text from the document and add their content to the search index.

Searching file metadata

If you want to include additional information about the document, you can directly associate [metadata](#) with the blobs, without using a separate store. The built-in [Blob Storage search indexer](#) can even read this [metadata](#) and place it in the search index. This enables users to search for metadata along with the file content. However, the [amount of metadata is limited to 8 KB per blob](#), so the amount of information that you can place on each blob is fairly small. You might choose to store only the most critical information directly on the blobs. In this scenario, only the document's *author* is stored on the blob.

To overcome this storage limitation, you can place additional metadata in another [data source that has a supported indexer](#), like [Table Storage](#). You can add the document type, business impact, and other metadata values as separate columns in the table. If you configure the built-in [Table Storage indexer](#) to target the same search index as the blob indexer, the blob and table storage metadata is combined for each document in the search index.

Using multiple data sources for a single search index

To ensure that both indexers point to the same document in the search index, the [document key in the search index](#) is set to a unique identifier of the file. This unique identifier is then used to refer to the file in both data sources. The blob indexer uses the `metadata_storage_path` as the [document key, by default](#). The `metadata_storage_path` property stores the full URL of the

file in Blob Storage, for example,

[https://contoso.blob.core.windows.net/files/paper/Resilience in Azure.pdf](https://contoso.blob.core.windows.net/files/paper/Resilience%20in%20Azure.pdf). The indexer performs Base64 encoding on the value to ensure that there are no invalid characters in the document key. The result is a unique document key, like `aHR0cHM6...mUucGRm0`.

If you add the `metadata_storage_path` as a column in Table Storage, you know exactly which blob the metadata in the other columns belongs to, so you can use any `PartitionKey` and `RowKey` value in the table. For example, you could use the blob container name as the `PartitionKey` and the Base64-encoded full URL of the blob as the `RowKey`, ensuring that there are no [invalid characters in these keys](#) either.

You can then use a [field mapping](#) in the table indexer to map the `metadata_storage_path` column (or another column) in Table Storage to the `metadata_storage_path` document key field in the search index. If you apply the [base64Encode function](#) on the field mapping, you end up with the same document key (`aHR0cHM6...mUucGRm0` in the earlier example), and the metadata from Table Storage is added to the same document that was extracted from Blob Storage.

! Note

The table indexer documentation states that [it concatenates the PartitionKey and RowKey as the document key, by default](#). Because you're already relying on the document key as configured by the blob indexer (which is the Base64-encoded full URL of the blob), creating a field mapping to ensure that both indexers refer to the same document in the search index is appropriate and supported for this scenario.

Alternatively, you can map the `RowKey` (which is set to the Base64-encoded full URL of the blob) to the `metadata_storage_path` document key directly, without storing it separately and Base64-encoding it as part of the field mapping. However, keeping the unencoded URL in a separate column clarifies which blob it refers to and allows you to choose any partition and row keys without affecting the search indexer.

Potential use cases

This scenario applies to applications that require the ability to search for documents based on their content and additional metadata.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more

information, see [Microsoft Azure Well-Architected Framework](#).

Reliability

Reliability ensures that your application can meet the commitments you make to your customers. For more information, see [Design review checklist for Reliability](#).

Azure AI Search provides a [high service-level agreement \(SLA\)](#) for *reads* (querying) if you have at least two [replicas](#). It provides a high service-level agreement (SLA) for *updates* (updating the search indexes) if you have at least three replicas. You should therefore provision at least two replicas if you want your users to be able to search reliably, and three if actual changes to the index also need to be high-availability operations.

[Azure Storage always stores multiple copies of your data](#) to help protect it against planned and unplanned events. Azure Storage provides additional redundancy options for replicating data across regions. These safeguards apply to data in blob and table storage.

Security

Security provides assurances against deliberate attacks and the abuse of your valuable data and systems. For more information, see [Design review checklist for Security](#).

Azure AI Search provides [robust security controls](#) that help you implement network security, authentication and authorization, data residency and protection, and administrative controls that help you maintain security, privacy, and compliance.

Whenever possible, use [Microsoft Entra authentication](#) to provide access to the search service itself, and connect your search service to other Azure resources (like blob and table storage in this scenario) by using a [managed identity](#).

You can connect from the search service to the storage account by using a [private endpoint](#). When you use a private endpoint, the indexers can use a private connection without requiring the blob and table storage to be accessible publicly.

Cost Optimization

Cost Optimization is about reducing unnecessary expenses and improving operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

For information about the costs of running this scenario, see this preconfigured [estimate in the Azure pricing calculator](#). All the services described here are configured in this estimate. The estimate is for a workload that has a total document size of 20 GB in Blob Storage and 1 GB of metadata in Table Storage. Two search units are used to satisfy the SLA for read purposes, as

described in the [Reliability](#) section of this article. To see how the pricing would change for your particular use case, change the appropriate variables to match your expected usage.

If you review the estimate, you can see that the cost of blob and table storage is relatively low. Most of the cost is incurred by Azure AI Search, because it performs the actual indexing and compute for running search queries.

Deploy this scenario

To deploy this example workload, see [Indexing file contents and metadata in Azure AI Search](#). You can use this sample to:

- Create the required Azure services.
- Upload a few sample documents to Blob Storage.
- Populate the *author* metadata value on the blob.
- Store the *document type* and *business impact* metadata values in Table Storage.
- Create the indexers that maintain the search index.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Jelle Druyts](#) | Principal Customer Experience Engineer

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Get started with Azure AI Search](#)
- [Increase relevancy using semantic search in Azure AI Search](#)
- [Security filters for trimming results in Azure AI Search](#)
- [Tutorial: Index from multiple data sources using the .NET SDK](#)

Related resource

- [Choose a search data store in Azure](#)

Automate video analysis by using Azure Machine Learning and Azure AI Vision

Azure Machine Learning

Azure AI services

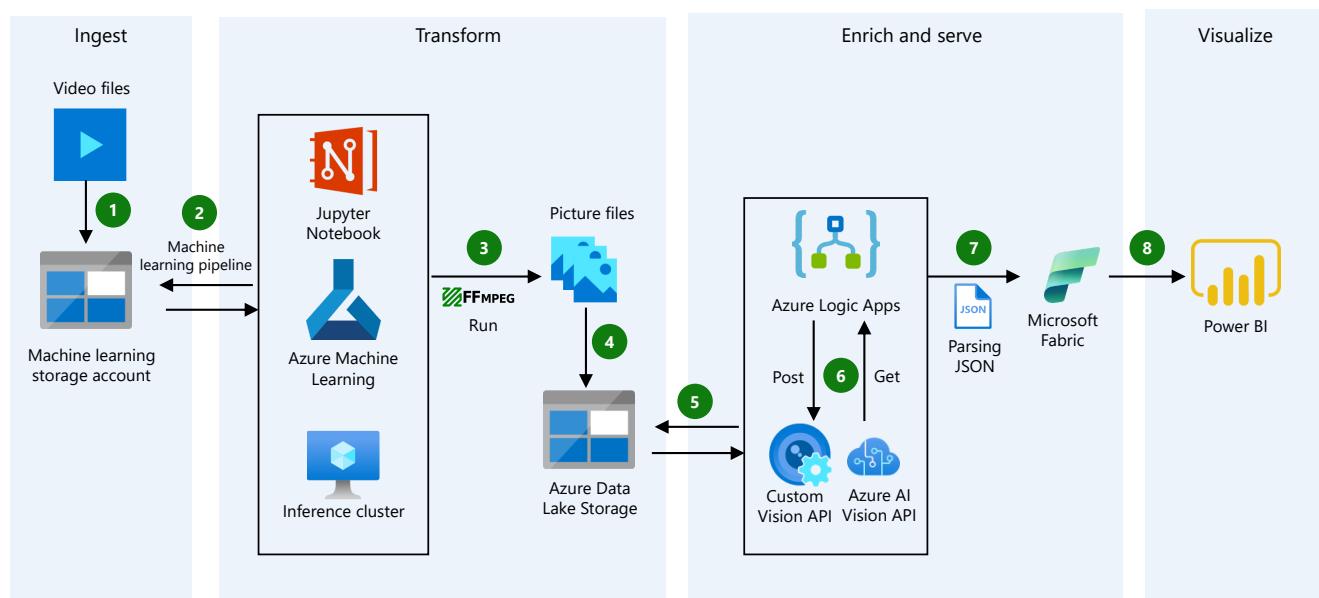
Azure Logic Apps

Azure Data Lake Storage

Microsoft Fabric

This article describes an architecture that replaces manual video analysis with an automated machine learning process. The automated process often produces more accurate results.

Architecture



Microsoft Azure



Download a [PowerPoint file](#) of this architecture.

The FFmpeg and Jupyter Notebook logos are trademarks of their respective companies. No endorsement is implied by the use of these marks.

Workflow

The following workflow corresponds to the previous diagram:

1. A collection of MP4 video footage is uploaded to Blob Storage, which serves as the initial storage location. The files are then processed in a machine learning storage account before frame extraction. Ideally, the videos go into a raw container.
2. A preconfigured pipeline in Machine Learning detects that video files are uploaded to the container and initiates an inference cluster to start separating the video footage into frames.

3. FFmpeg, which is an open-source tool, decodes the video and extracts individual frames as image files. You can set up how many frames per second are extracted, the quality of the extraction, and the format of the image file. The format can be JPG or PNG.
4. The inference cluster sends the images to Azure Data Lake Storage.
5. A preconfigured logic app that monitors Data Lake Storage detects that new images are being uploaded. It starts a workflow.
6. The logic app calls a pretrained Azure AI Custom Vision model to identify objects, features, or qualities in the images. It might also call a computer vision model that uses optical character recognition (OCR) to identify textual information in the images.
7. Results arrive in JSON format. The logic app parses the results to create key-value pairs. You can store those pairs in Fabric Data Warehouse, which is a managed analytical database that supports atomicity, consistency, isolation, and durability (ACID) transactions. It uses the open Delta Parquet format and integrates natively with OneLake.
8. Power BI provides data visualization.

Components

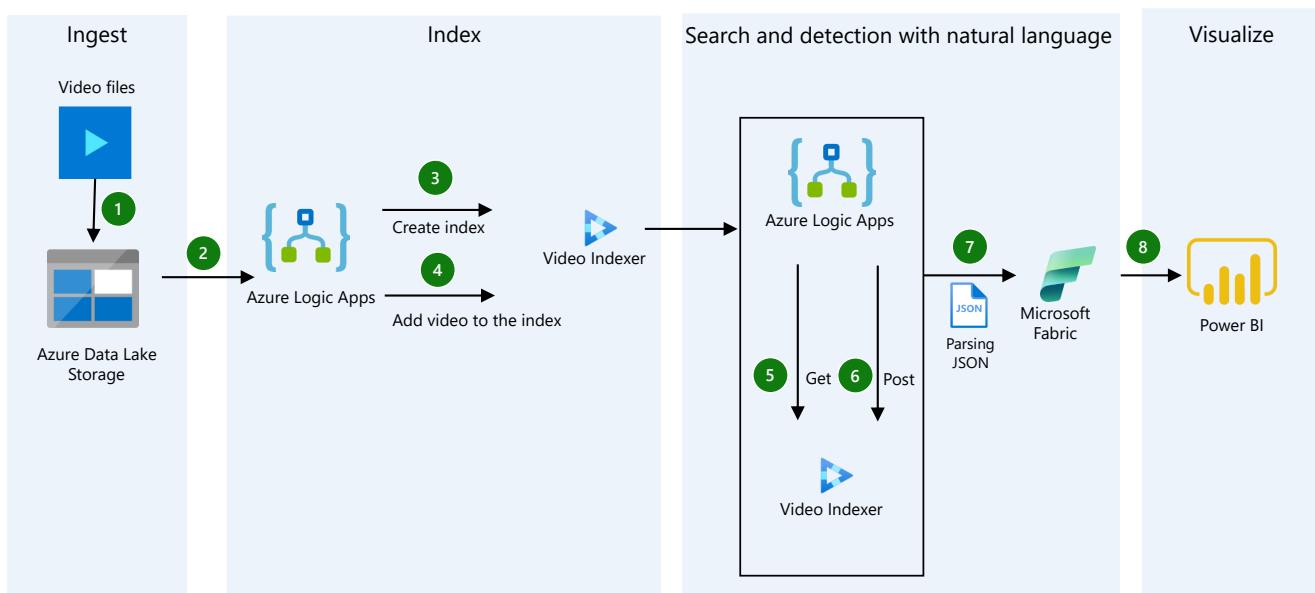
- **Blob Storage** is a cloud-based solution designed to store objects for cloud-native applications and machine learning workloads. In this architecture, it stores the uploaded raw video footage that serves as the input for the automated video analysis pipeline.
- **Machine Learning** is an enterprise-grade service that supports the end-to-end machine learning life cycle. In this architecture, it manages the inference cluster and orchestrates the process of breaking down videos into frames for further analysis.
- **Data Lake Storage** is a scalable, secure, and cost-effective cloud storage solution for analytics workloads. In this architecture, it stores the extracted video frames for downstream image analysis and processing.
- **Azure AI Vision** is part of **Azure AI services**. It provides tools to extract information from images. In this architecture, it analyzes the extracted frames and identifies objects and features. It can also retrieve text via OCR.
- **Custom Vision** is a cloud-based AI service that you can use to customize and embed advanced computer vision image analysis for your specific domains. In this architecture, it identifies domain-specific objects or qualities in the extracted video frames.
- **Azure Logic Apps** is a cloud-based service that automates workflows by connecting apps and data across environments. It provides a way to access and process data in real time.

In this architecture, it monitors storage locations, initiates analysis workflows, processes results, and coordinates the movement and transformation of data.

- **Microsoft Fabric** is an end-to-end unified analytics platform that streamlines integration of data workflows across data engineering, data integration, warehousing, real-time analytics, and business intelligence (BI). In this architecture, after Logic Apps parses the JSON results, the data is stored and analyzed in **Data Warehouse** for governed analytics before visualization in Power BI.
- **Power BI** is a collection of software services, apps, and connectors that work together to provide visualizations of your data. In this architecture, Power BI provides dashboards and reports that visualize the results of the automated video analysis to enable insights and decision-making.

Alternatives

If you don't need to call a pretrained object detection custom model, use the following architecture that relies on Microsoft Azure AI Video Indexer. This service removes the decomposition of video into frames and the use of custom code to parse through the ingestion process. This approach is more direct if your use case involves detecting common objects or entities in a video.



Download a [PowerPoint file](#) of this architecture.

Alternative workflow

The following workflow corresponds to the previous diagram:

1. A collection of MP4 video footage is uploaded to Blob Storage, which is used for video upload before indexing and further processing in Data Lake Storage.
2. A preconfigured logic app monitors Blob Storage, detects that new videos are being uploaded, and starts a workflow.
3. The logic app calls the Video Indexer API to create an index.
4. The logic app calls the Video Indexer API to add video documents to the index.
5. A preconfigured logic app monitors the ingestion to check when the indexing is complete.
6. The logic app uses the Video Indexer API to perform natural language searches and detect objects, features, or attributes in images.
7. Results are received in JSON format. The logic app parses the results and creates key-value pairs. You can store the results in SQL database in Fabric.
8. Power BI provides data visualization.

Alternative components

- [SQL database in Fabric](#) is a managed SQL database service designed to support AI-driven workloads securely and efficiently. In this architecture, it stores information about the videos retrieved from the Video Indexer API.
- [Video Indexer](#) is a service that enables direct analysis of video files for object, feature, and attribute detection. It supports natural language search over indexed video content. In this architecture, Video Indexer lets you retrieve structured information from videos without manual frame extraction or custom code.

Scenario details

Many industries record video footage to detect the presence or absence of a specific object or entity or to classify objects or entities. Video monitoring and analyses are typically performed manually. These processes are often monotonous and prone to errors, especially for tasks that are difficult for the human eye. You can automate these processes by using AI and machine learning.

A video recording can be separated into individual frames so that different technologies can analyze the images. An example of this technology is *computer vision*, which is the capability of

a computer to identify objects and entities in an image.

With computer vision, monitoring video footage becomes automatized, standardized, and potentially more accurate. A computer vision model can be trained. Depending on the use case, you can frequently get results that are at least as good as the results of the person who trained the model. You can achieve better results and adapt to changes in video data over time by using [machine learning operations](#) to continuously improve the model.

Potential use cases

This scenario is relevant for any business that analyzes videos. Consider the following sample use cases:

- **Agriculture:** Monitor and analyze crops and soil conditions over time. Farmers can record video footage for analysis by using drones or unmanned aerial vehicles (UAVs).
- **Environmental sciences:** Analyze aquatic species to learn where they're located and how they evolve. Environmental researchers can navigate the shoreline and record video footage by attaching underwater cameras to boats. They can analyze the video footage to understand species migrations and how species populations change over time.
- **Traffic control:** Classify vehicles into categories, like SUV, car, truck, and motorcycle. Use that information to plan traffic control. Closed-circuit television (CCTV) in public locations can provide video footage. Most CCTV cameras record the date and time, which can be retrieved via OCR.
- **Quality assurance:** Monitor and analyze quality control in a manufacturing facility. By installing cameras on the production line, you can train a computer vision model to detect anomalies.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Well-Architected Framework](#).

Reliability

Reliability helps ensure that your application can meet the commitments that you make to your customers. For more information, see [Design review checklist for Reliability](#).

A reliable workload is resilient and available. *Resiliency* is the ability of the system to recover from failures and continue to function. The goal of resiliency is to return the application to a fully functioning state after a failure occurs. *Availability* is a measure of whether your users can access your workload when they need to.

For the availability guarantees of the Azure services in this solution, see [Service-level agreements \(SLAs\) for online services](#).

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

Consider the following guidelines for optimizing costs:

- Use the pay-as-you-go strategy for your architecture, and [scale out](#) as needed rather than investing in large-scale resources at the start.
- Consider opportunity costs in your architecture and the trade-offs between being a first-mover and adopting a fast-follow strategy. Use the [Azure pricing calculator](#) to estimate the initial cost and operational costs.
- Establish [policies, budgets, and controls](#) that set cost limits for your solution.

Operational Excellence

Operational Excellence covers the operations processes that deploy an application and keep it running in production. For more information, see [Design review checklist for Operational Excellence](#).

Deployments need to be reliable and predictable. Consider the following guidelines:

- Automate deployments to reduce the chance of human error.
- Implement a fast, repeatable deployment process to ensure timely release of new features and bug fixes.
- Roll back or roll forward quickly if an update causes problems.

Performance Efficiency

Performance Efficiency refers to your workload's ability to scale to meet user demands efficiently. For more information, see [Design review checklist for Performance Efficiency](#).

The appropriate use of scaling and the implementation of platform as a service (PaaS) offerings that have built-in scaling are the main ways to achieve performance efficiency.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal authors:

- [Oscar Shimabukuro Kiyan ↗](#) | Senior Cloud Solutions Architect – Data & AI
- [Han Wang ↗](#) | Cloud Solutions Architect – Data & AI

Other contributors:

- [Rodrigo Rodríguez ↗](#) | Senior Cloud Solution Architect, AI & Quantum

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Introduction to Azure Storage](#)
- [What is Machine Learning?](#)
- [What is Power BI embedded analytics?](#)

Extract and analyze call center data

Azure Blob Storage

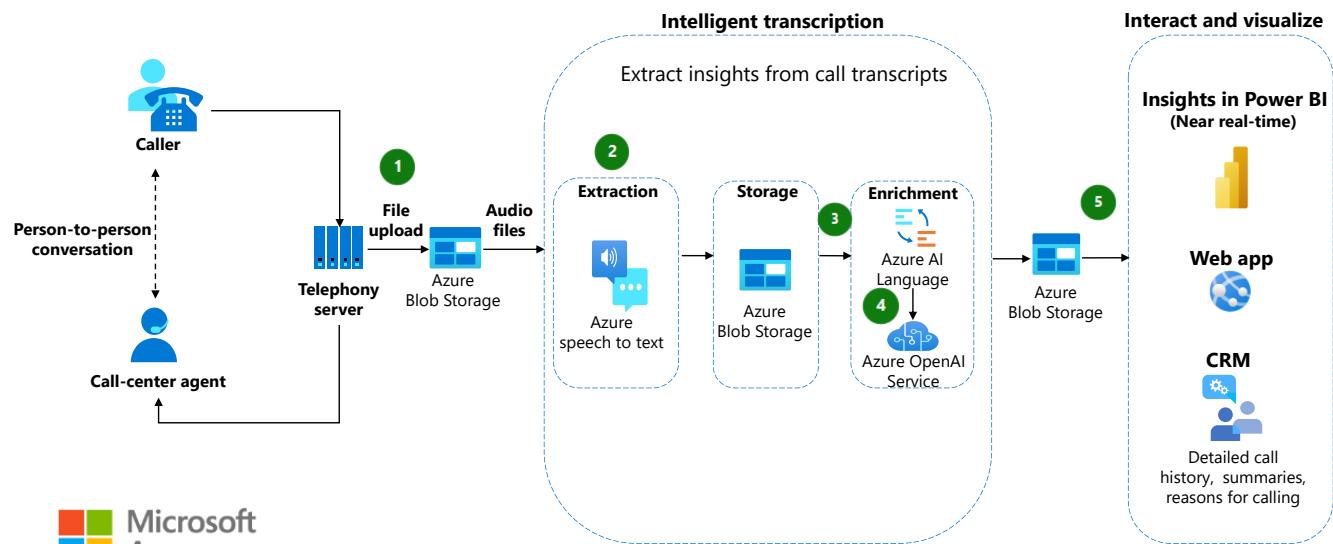
Azure AI Speech

Azure AI services

Power BI

This article describes how to extract insights from customer conversations at a call center by using Azure AI services and Azure OpenAI Service. Use these services to improve your customer interactions and satisfaction by analyzing call intent and sentiment, extracting key entities, and summarizing call content.

Architecture



Download a [PowerPoint file](#) of this architecture.

Dataflow

1. A phone call between an agent and a customer is recorded and stored in Azure Blob Storage. Audio files are uploaded to an Azure Storage account via a supported method, such as the UI-based tool, [Azure Storage Explorer](#), or a [Storage SDK or API](#).
2. An Azure function is configured with one of the following triggers to start the intelligent transcription process:
 - **Timer trigger:** Configure a time-based trigger to process a batch of audio files accumulated over a specified time period.
 - **Blob trigger:** Configure a blob trigger to initiate intelligent transcription as soon as an audio file is uploaded to the blob container.

3. The Azure function will trigger an Azure App Service which will execute the following steps in sequence:
 - a. Call the Azure AI Speech to transcribe the files.
 - b. Optionally, save this raw file in Azure blob storage for future reference.
 - c. Pass the raw data to the Azure AI Language service to [detect and redact personal data](#) in the transcript.
 - d. Send the redacted data to the Azure OpenAI service to perform various post call analytics like understand the intent and sentiment of the call, extract entities, or [summarize the conversation](#) to evaluate the effectiveness of the call.
 - e. Store the processed output in Azure Storage for visualization or consumption by downstream applications for further processing.
 4. [Power BI](#) can be used to visualize the post call analytics on different criteria as required by the business use case. You can also store this output in a customer relationship management (CRM), so agents have contextual information about why the customer called and can quickly solve potential problems. This process is fully automated, which saves the agents time and effort.
- ## Components
- [Blob Storage](#) is the object storage solution for raw files in this scenario. Blob Storage supports libraries for languages like .NET, Node.js, and Python. Applications can access files on Blob Storage via HTTP or HTTPS. Blob Storage has [hot, cool, and archive access tiers](#) for storing large amounts of data, which optimizes cost.
 - [Azure OpenAI](#) provides access to the Azure OpenAI language models, including GPT-3, Codex, and the embeddings model series, for content generation, summarization, semantic search, and natural language-to-code translation. You can access the service through REST APIs, Python SDK, or the web-based interface in the [Azure OpenAI studio](#).
 - [Azure AI Speech](#) is an AI-based API that provides speech capabilities like speech-to-text, text-to-speech, speech translation, and speaker recognition. This architecture uses the Azure AI Speech batch transcription functionality.
 - [Azure AI Language](#) consolidates the Azure natural-language processing services. For more information about prebuilt and customizable options, see [Azure AI Language available features](#).

- [Language Studio](#) provides a UI for exploring and analyzing AI services for language features. Language Studio provides options for building, tagging, training, and deploying custom models.
- [Power BI](#) is a software-as-a-service (SaaS) that provides visual and interactive insights for business analytics. It provides transformation capabilities and connects to other data sources.

Alternatives

Depending on your scenario, you can add the following workflows.

- Perform [conversation summarization](#) by using the prebuilt model in Azure AI Language.
- Azure also offers Speech Analytics which provides the entire orchestration for post call analytics in batch.

Scenario details

This solution uses Azure AI Speech to Text to convert call-center audio into written text. Azure AI Language redacts sensitive information in the conversation transcription. Azure OpenAI extracts insights from customer conversation to improve call center efficiency and customer satisfaction. Use this solution to process transcribed text, recognize and remove sensitive information, and perform analytics on the extractions like reason for the call, resolution provided or not, sentiment of the call, listing product /service offering based on the number of queries/customer complaints, and so on. Scale the services and the pipeline to accommodate any volume of recorded data.

Potential use cases

This solution provides value to organizations across multiple industries that have customer support agents. The post call analytics can help improve the company's products and services, and the effectiveness of the customer support systems. The solution applies to any organization that records conversations, including customer-facing agents, internal call centers, or support desks.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Well-Architected Framework](#).

Reliability

Reliability helps ensure that your application can meet the commitments that you make to your customers. For more information, see [Design review checklist for Reliability](#).

- Find the availability service-level agreement (SLA) for each component in [SLAs for online services ↗](#).
- To design high-availability applications with Storage accounts, see the [configuration options](#).
- To ensure resiliency of the compute services and datastores in this scenario, use failure mode for services like Azure Functions and Storage. For more information, see [Reliability guides by service](#).

Security

Security provides assurances against deliberate attacks and the misuse of your valuable data and systems. For more information, see [Design review checklist for Security](#).

- Implement data protection, identity and access management, and network security recommendations for [Blob Storage](#), [AI services](#), and [Azure OpenAI](#).
- [Configure AI services virtual networks](#).

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

The total cost of this solution depends on the pricing tier of your services. Factors that can affect the price of each component are:

- The number of documents that you process.
- The number of concurrent requests that your application receives.
- The size of the data that you store after processing.
- Your deployment region.

For more information, see the following resources:

- [Azure OpenAI pricing ↗](#)
- [Blob Storage pricing ↗](#)
- [Azure AI Language pricing ↗](#)
- [Azure Machine Learning pricing ↗](#)

Use the [Azure pricing calculator ↗](#) to estimate your solution cost.

Performance Efficiency

Performance Efficiency refers to your workload's ability to scale to meet user demands efficiently. For more information, see [Design review checklist for Performance Efficiency](#).

When high volumes of data are processed, it can expose performance bottlenecks. To ensure proper performance efficiency, understand and plan for the [scaling options](#) to use with the [AI services autoscale feature](#).

The batch speech API is designed for high volumes, but other AI services APIs might have request limits, depending on the subscription tier. Consider containerizing AI services APIs to avoid slowing down large-volume processing. Containers provide deployment flexibility in the cloud and on-premises. Mitigate side effects of new version rollouts by using containers. For more information, see [Container support in AI services](#).

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- Dixit Arora | Senior Customer Engineer, ISV DN CoE
- [Jyotsna Ravi](#) | Principal Customer Engineer, ISV DN CoE

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- [What is Azure AI Speech?](#)
- [What is Azure OpenAI?](#)
- [What is Azure Machine Learning?](#)
- [Introduction to Blob Storage](#)
- [What is Azure AI Language?](#)
- [Introduction to Azure Data Lake Storage Gen2](#)
- [What is Power BI?](#)
- [Ingestion Client with AI services](#)
- [Post-call transcription and analytics](#)

Related resource

- [Create custom language and acoustic models](#)

Design a secure research environment for regulated data

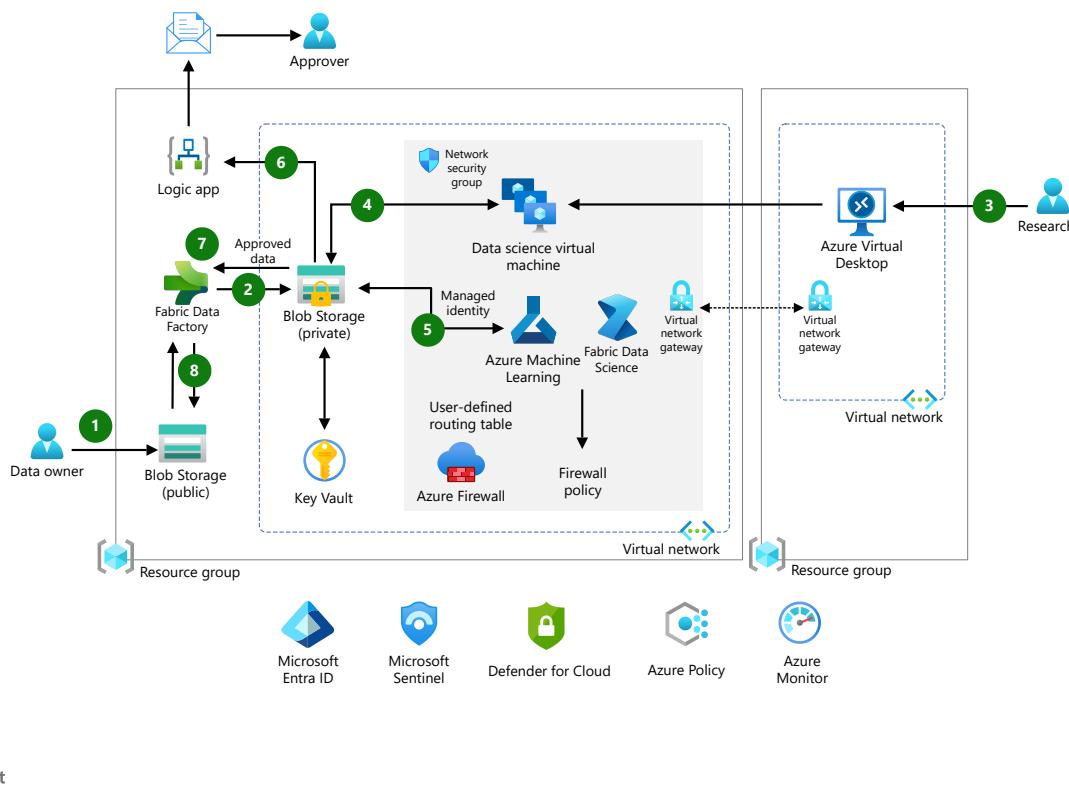
Azure Data Science Virtual Machines

Azure Machine Learning

Microsoft Fabric

This article describes a secure research environment that provides researchers access to sensitive data that requires a high level of control and protection. The architecture supports organizations that must adhere to regulatory compliance or other strict security requirements.

Architecture



Download a [Visio file](#) of this architecture.

Data flow

The following data flow corresponds to the previous diagram:

1. Data owners upload datasets into a public Azure Blob Storage account. They use Microsoft-managed keys to encrypt the data.
2. **Fabric Data Factory** uses a trigger to copy the uploaded dataset to a specific location or import path on another storage account that has security controls. You can only reach the storage account through a private endpoint or trusted workspace access. A service

principal that has limited permissions can also access the account. Data Factory deletes the original copy, which makes the dataset immutable.

3. Researchers access the secure environment through a streaming application by using [Azure Virtual Desktop](#) as a privileged jump box.
4. The secure storage account presents the dataset to the data science virtual machines (VMs) that you set up in a secure network environment for research work. Most data preparation occurs on those VMs.
5. The secure environment includes Azure Machine Learning and [Fabric Data Science](#). They can access the dataset through a private endpoint. You can use these platforms to train, deploy, automate, and manage machine learning models. At this stage, you can create models that meet regulatory guidelines. To de-identify all model data, remove personal information.
6. Models or de-identified data are saved to a separate location on the secure storage account, known as the *export path*. When you add new data to the export path, you trigger a logic app. In this architecture, the logic app runs outside the secure environment because it doesn't receive data. Its only function is to send notifications and start the manual approval process.

The logic app starts an approval process by requesting a review of data that's queued for export. The manual reviewers help ensure that sensitive data isn't exported. After the review process, reviewers either approve or deny the data.

 **Note**

If you don't need to approve data exports, you can skip the logic app step.

7. If reviewers approve the de-identified data, the system sends it to Data Factory.
8. Data Factory moves the data to the public storage account in a separate container so that external researchers can access their exported data and models. Alternatively, you can set up another storage account in a lower security environment.

Components

This architecture consists of several Azure services that scale resources according to your needs. The following sections describe these services and their roles.

Core workload components

The following core components move and process research data:

- **Azure data science VMs** are VMs that you configure with tools for data analytics and machine learning. In this architecture, they provide researchers with dedicated, secure compute resources for data preparation, analysis, and model training within the isolated environment. Data science VMs provide specific packages or tools, such as Matrix Laboratory (MATLAB) or Statistical Analysis System (SAS), that platform as a service (PaaS) environments can't support. For security and ease of use, choose Machine Learning and other PaaS options when supported.
- **Machine Learning** is a service that trains, deploys, automates, and manages machine learning models. In this architecture, it facilitates model development and orchestration while maintaining security controls over data access and compute resources. It can also manage the allocation and use of machine learning compute resources. Machine Learning provides the preferred environment to run Jupyter notebooks during development.
- **Machine Learning compute** is a cluster of nodes that can train and test machine learning and AI models. In this architecture, it provides automatically scalable, secure, and isolated compute resources for research. You can deploy Visual Studio Code (VS Code) as a streaming application from Virtual Desktop and connect it to the Machine Learning compute for an alternative development environment.
- **Blob Storage** is an object storage solution that stores unstructured data in the cloud. In this architecture, it serves as the primary storage solution, and it has two instances. The public instance temporarily stores the data that data owners upload. It stores de-identified data after it models the data in a separate container. The private instance receives the training and test datasets from Machine Learning. The training scripts use those datasets. The system mounts storage as a virtual drive onto each node of a Machine Learning compute cluster.
- **Fabric** is an analytical platform for big data and pipelines that provides data integration and extract, transform, load (ETL) capabilities. It serves as a preferred service to run Apache Spark workloads. In this architecture, Fabric enables advanced analytics and data integration for research datasets that you can access through secure, private endpoints.
- **Data Factory** is a cloud-based data integration service within Fabric that orchestrates and operationalizes data movement and transformation workflows. In this architecture, it moves data between storage accounts that have different security levels, enforces separation of duties, and manages data flows throughout the secure environment.
- **Virtual Desktop** is a desktop and app virtualization service that runs in the cloud. In this architecture, it acts as a jump box that provides access to the resources in the secure

environment. Researchers can use Virtual Desktop to connect to data science VMs through streaming applications and a full desktop as needed.

Alternatively, you can use [Azure Bastion](#), but understand the security control differences between the two options. Virtual Desktop has the following advantages:

- Stream applications like VS Code to run notebooks on machine learning compute resources
- Limit copy, paste, and screen captures
- Support Microsoft Entra authentication to data science VMs
- [Azure Logic Apps](#) is a service that automates workflows and integrates apps, data, systems, and services across enterprises or organizations. In this architecture, it manages the trigger and release portions of the manual approval process.

Posture management components

The following components continuously monitor the posture of the workload and its environment. Use these components to discover risks and immediately mitigate them.

- [Microsoft Defender for Cloud](#) is a service that evaluates the overall security posture of the implementation and provides an attestation mechanism for regulatory compliance. In this architecture, it helps detect problems early, instead of during audits or assessments. Use features like the secure score and compliance score to track progress. These scores help check compliance.
- [Microsoft Sentinel](#) is a security information and event management (SIEM) solution and a security orchestration, automation, and response (SOAR) solution. In this architecture, it centralizes logs, detects threats, and automates security responses for the research environment. You can centrally view logs and alerts from various sources. Take advantage of advanced AI and security analytics to detect, investigate, prevent, and respond to threats. This capability provides valuable security insights to help ensure that traffic and activities associated with the workspace meet your expectations.
- [Azure Monitor](#) is a monitoring solution that collects, analyzes, and responds to telemetry data from cloud and on-premises environments. In this architecture, it collects and visualizes metrics, activity logs, and diagnostics to support operational monitoring and incident detection. Management tools, like tools in Defender for Cloud, also push log data to Azure Monitor.

Governance components

- [Azure Policy](#) is a governance tool that enforces organizational standards and assesses compliance at scale. In this architecture, it helps ensure that resources and workloads adhere to security and configuration policies.

Alternatives

- This solution uses Data Factory to move data to a public storage account in a separate container so that external researchers can access their exported data and models. Alternatively, you can use Data Factory to transfer data to a public storage account in a separate container or set up another storage account in a lower security environment for the same purpose.
- This solution uses Virtual Desktop as a jump box to gain access to the resources in the secure environment by providing streaming applications and a full desktop. Alternatively, you can use Azure Bastion, but Virtual Desktop has advantages. These advantages include the ability to stream an app, to limit copy and paste capabilities and screen captures, and to support Microsoft Entra authentication. Also consider configuring a point-to-site virtual private network (VPN) to support local offline training. This VPN helps reduce the cost of multiple VMs for workstations.
- To secure data at rest, this solution encrypts all Azure Storage accounts with Microsoft-managed keys by using strong cryptography. Alternatively, you can use customer-managed keys. You must store the keys in a managed key store.

Scenario details

This scenario combines regulated and private data that individuals must access but aren't allowed to store or transmit. The following conditions apply:

- Data scientists outside your organization need full access to the data to train and export their models, but no proprietary or protected data can leave the environment.
- You must isolate access. Even the data owners and custodians can't access the data after it's uploaded into the environment.
- You must require an audit trail for exports to ensure that only the models are transferred out of the environment.

Potential use cases

This architecture was originally created for higher education research institutions that have Health Insurance Portability and Accountability Act (HIPAA) requirements. You can use this

design in any industry that requires data isolation for research purposes. Consider the following examples:

- Industries that process regulated data in accordance with National Institute of Standards and Technology (NIST) requirements
- Medical centers that collaborate with internal or external researchers
- Banking and finance industries

Follow the guidance in this article to maintain full control of your research data, maintain separation of duties, and meet strict regulatory compliance standards. This approach also facilitates collaboration among key roles in a research-oriented environment, like data owners, researchers, and approvers.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Well-Architected Framework](#).

Reliability

Reliability helps ensure that your application can meet the commitments that you make to your customers. For more information, see [Design review checklist for Reliability](#).

Most research solutions consist of temporary workloads that don't need to remain available for extended periods. This architecture uses a single-region deployment with availability zones. If your business requirements demand higher availability, replicate this architecture in multiple regions. Add components, like a global load balancer and distributor, to route traffic to those regions. As part of your recovery strategy, use Azure VM Image Builder to capture and create a copy of the customized base image.

Security

Security provides assurances against deliberate attacks and the misuse of your valuable data and systems. For more information, see [Design review checklist for Security](#).

The main objective of this architecture is to provide a secure and trusted research environment that strictly limits data exfiltration from the secure area.

Network security

Set up Azure resources in a secure environment. These resources store, test, and train research datasets. The environment resides in an Azure virtual network that has network security group (NSG) rules to restrict access. These rules apply to the following areas:

- Inbound and outbound access to the public internet and within the virtual network.
- Access to and from specific services and ports. For example, this architecture blocks all port ranges except the ones required for Azure services, like Azure Monitor. For a full list of service tags and the corresponding services, see [Virtual network service tags](#).

Access from the virtual network that includes Virtual Desktop is restricted to approved access methods on specific ports, but all other traffic is denied. Compared to this environment, the other virtual network that includes Virtual Desktop is relatively open.

The main Blob Storage instance in the secure environment isn't exposed to the public internet. You can access it only within the virtual network through [private endpoint connections](#) and Storage firewalls. Use these controls to limit the networks that clients can use to connect to file shares in Azure Files.

This architecture uses credential-based authentication for the main data store in the secure environment. In this setup, a key vault stores the connection information, like the subscription ID and token authorization. Alternatively, you can create identity-based data access, where you use your Azure account to confirm whether you have access to Storage, without saving authentication credentials. For more information, see [Create data stores](#).

The compute cluster can communicate only within the virtual network by using the Azure Private Link ecosystem and service or private endpoints. It doesn't use public IP addresses for communication. Enable the **No public IP** setting. For more information about this feature, see [Compute instance and cluster or serverless compute with no public IP address](#).

The secure environment uses Machine Learning compute to access the dataset through a private endpoint. You can also configure Azure Firewall to control both inbound and outbound access to Machine Learning compute, which resides in a machine learning workspace. For more information, see [Configure inbound and outbound network traffic](#).

For more information, see [Secure a Machine Learning service environment](#).

For Azure services that you can't configure effectively by using private endpoints, or to provide stateful packet inspection, consider using Azure Firewall or a non-Microsoft network virtual appliance (NVA).

Identity management

This architecture implements multiple layers of identity-based security controls. You can access Blob Storage through Azure role-based access control (Azure RBAC). Virtual Desktop supports Microsoft Entra authentication to data science VMs, which adds an extra security layer for researcher access.

Data Factory uses [trusted workspace access](#) to securely connect to data in Blob Storage accounts. This approach uses the workspace's managed identity to bypass firewall restrictions and access protected storage without requiring public network exposure. Data science VMs also use managed identity for remediation tasks to ensure secure operations across the Fabric environment.

Data security

To secure data at rest, Microsoft-managed keys encrypt all Storage accounts by using strong cryptography.

Alternatively, you can use customer-managed keys. You must store the keys in a managed key store. In this architecture, you deploy Azure Key Vault in the secure environment to store secrets like encryption keys and certificates. Resources in the secure virtual network access Key Vault through a private endpoint.

Governance considerations

Enable Azure Policy to enforce standards and provide automated remediation to make resources compliant with specific policies. You can apply the policies to a project subscription or at a management group level, either as a single policy or as part of a regulatory initiative.

For example, in this architecture, Azure machine configuration applies to all in-scope VMs. The policy can audit operating systems and machine configuration for the data science VMs.

VM image

The data science VMs run customized base images. To build the base image, use technologies like VM Image Builder. You can create a repeatable image to deploy when needed.

The base image might need updates, like extra binaries. Upload those binaries to the public Blob Storage instance. They should flow through the secure environment, similar to how data owners upload the datasets.

Cost Optimization

Cost Optimization focuses on ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

The cost of data science VMs depends on the underlying VM series. The workload is temporary, so use the Consumption plan for the logic app resource. To estimate costs based on the estimated sizing of resources that you need, use the [Azure pricing calculator](#). Shut down the environment when it's not in use to help optimize costs and improve security.

Performance Efficiency

Performance Efficiency refers to your workload's ability to scale to meet user demands efficiently. For more information, see [Design review checklist for Performance Efficiency](#).

Choose the appropriate size and type of the data science VMs for the style of work that they do. This architecture supports a single research project. To achieve scalability, adjust the size and type of the VMs and choose compute resources that Machine Learning supports.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal author:

- [Clayton Barlow](#) | Senior Azure Specialist

Other contributor:

- [Tincy Elias](#) | Senior Cloud Solution Architect

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- [What is the data science VM for Linux and Windows?](#)
- [What is Machine Learning?](#)
- [Introduction to Data Science](#)
- [What are compute targets in Machine Learning?](#)
- [Introduction to Blob Storage](#)
- [What is Data Factory in Fabric?](#)
- [What is Virtual Desktop?](#)
- [Defender for Cloud documentation](#)
- [What is Microsoft Sentinel SIEM?](#)

- Azure Monitor overview
- What is Azure Policy?
- Understand Azure machine configuration

Related resources

- [Compare Microsoft machine learning products and technologies](#)
- [Use the many-models architecture approach to scale machine learning models](#)

Compare Microsoft machine learning products and technologies

Article • 01/29/2025

Learn about the machine learning products and technologies from Microsoft. Compare options to help you choose how to most effectively build, deploy, and manage your machine learning solutions.

Cloud-based machine learning products

The following options are available for machine learning in the Azure cloud.

[+] Expand table

| Cloud option | Description | Features and uses |
|--|--|--|
| Azure Machine Learning | Managed platform for machine learning | Use a pretrained model, or train, deploy, and manage models on Azure by using Python and a CLI. Machine Learning includes features like automated machine learning (AutoML), prompt flow, model catalog, and MLflow integration. You can track and understand model performance during the production stage. |
| Microsoft Fabric | Unified analytics platform | Manage the entire data lifecycle, from ingestion to insights, by using a comprehensive platform that integrates various services and tools for data professionals, including data engineers, data scientists, and business analysts. |
| Azure AI services | Prebuilt AI capabilities that are implemented through REST APIs and SDKs | Build intelligent applications by using standard programming languages. These languages call APIs that provide inferencing. Although you should ideally have machine learning and data science expertise, engineering teams that don't have these skills can also adopt this platform. |
| Azure SQL Managed Instance machine learning services | In-database machine learning for SQL | Train and deploy models inside SQL Managed Instance. |

| Cloud option | Description | Features and uses |
|---|--|--|
| Machine learning in Azure Synapse Analytics | Analytics service that uses machine learning | Train and deploy models inside Azure Synapse Analytics. |
| Azure Databricks | Apache Spark-based analytics platform | Build and deploy models and data workflows by integrating with open-source machine learning libraries and the MLflow platform. |

On-premises machine learning product

The following option is available for machine learning on-premises. On-premises servers can also run in a virtual machine (VM) in the cloud.

[\[\] Expand table](#)

| On-premises product | Description | Features and uses |
|--|--------------------------------------|--|
| SQL Server machine learning services | In-database machine learning for SQL | Train and deploy models inside SQL Server by using Python and R scripts. |

Development platforms and tools

The following development platforms and tools are available for machine learning.

[\[\] Expand table](#)

| Platform or tool | Description | Features and uses |
|---|---|--|
| Azure AI Foundry portal | Unified development environment for AI and machine learning scenarios | Develop, evaluate, and deploy AI models and applications. The Azure AI Foundry portal facilitates collaboration and project management across various Azure AI services. You can even use it as a common environment across multiple workload teams. |
| Azure Machine Learning studio | Collaborative, drag-and-drop tool for machine learning | Build, test, and deploy predictive analytics solutions by using minimal coding. Machine Learning studio supports a wide range of machine learning algorithms and AI models. It |

| Platform or tool | Description | Features and uses |
|--|--|---|
| | | provides tools for data preparation, model training, and evaluation. |
| Azure Data Science Virtual Machine | VM image that includes preinstalled data science tools | Use a preconfigured environment with tools like Jupyter, R, and Python to develop machine learning solutions on your own VMs. |
| Microsoft ML.NET | Open-source, cross-platform machine learning SDK | Develop machine learning solutions for .NET applications. |
| AI for Windows apps | Inference engine for trained models on Windows devices | Integrates AI capabilities into Windows applications by using components like Windows Machine Learning (WinML) and Direct Machine Learning (DirectML) for local, real-time AI model evaluation and hardware acceleration. |
| SynapseML | Open-source, distributed machine learning and microservices framework for Apache Spark | Create and deploy scalable machine learning applications for Scala and Python. |
| Machine learning extension for Azure Data Studio | Open-source and cross-platform machine learning extension for Azure Data Studio | Manage packages, import machine learning models, make predictions, and create notebooks to run experiments for your SQL databases. |

Azure Machine Learning

[Machine Learning](#) is a fully managed cloud service that you can use to train, deploy, and manage machine learning models at scale. It fully supports open-source technologies, so you can use tens of thousands of open-source Python packages, such as TensorFlow, PyTorch, and scikit-learn.

Rich tools, such as [compute instances](#), [Jupyter notebooks](#), or the [Azure Machine Learning for Visual Studio Code \(VS Code\) extension](#), are also available. The Machine Learning for VS Code extension is a free extension that allows you to manage your resources and model training workflows and deployments in VS Code. Machine Learning

includes features that automate model generation and tuning with ease, efficiency, and accuracy.

Use Python SDK, Jupyter notebooks, R, and the CLI for machine learning at cloud scale. If you want a low-code or no-code option, use [Designer](#) in the studio. Designer helps you easily and quickly build, test, and deploy models by using prebuilt machine learning algorithms. Additionally, you can integrate Machine Learning with Azure DevOps and GitHub Actions for continuous integration and continuous deployment (CI/CD) of machine learning models.

[+] [Expand table](#)

| Machine Learning feature | Description |
|--------------------------|--|
| Type | Cloud-based machine learning solution |
| Supported languages | <ul style="list-style-type: none">- Python- R |
| Machine learning phases | <ul style="list-style-type: none">- Data preparation- Model training- Deployment- MLOps or management- Responsible AI |
| Key benefits | <ul style="list-style-type: none">- Code-first (SDK) and studio and drag-and-drop designer web interface authoring options- Central management of scripts and run history, which makes it easy to compare model versions- Easy deployment and management of models to the cloud or edge devices- Scalable training, deployment, and management of machine learning models |
| Considerations | Requires some familiarity with the model-management model. |

Azure AI services

[AI services](#) is a comprehensive suite of prebuilt APIs that help developers and organizations create intelligent, market-ready applications rapidly. These services provide out-of-the-box and customizable APIs and SDKs that allow your apps to see, hear, speak, understand, and interpret user needs with minimal code. These capabilities make datasets or data science expertise to train models unnecessary. You can add intelligent features to your apps, such as:

- **Vision:** Includes object detection, face recognition, and optical character recognition. For more information, see [Azure AI Vision](#), [Azure AI Face](#), and [Azure AI Document Intelligence](#).
- **Speech:** Includes speech-to-text, text-to-speech, and speaker recognition capabilities. For more information, see [Speech service](#).
- **Language:** Includes translation, sentiment analysis, key phrase extraction, and language understanding. For more information, see [Azure OpenAI Service](#), [Azure AI Translator](#), [Azure AI Immersive Reader](#), [Bot Framework Composer](#), and [Azure AI Language](#).
- **Decision-making:** Detect unwanted content and make informed decisions. For more information, see [Azure AI Content Safety](#).
- **Search and knowledge:** Bring AI-powered cloud search and knowledge mining capabilities to your apps. For more information, see [Azure AI Search](#).

Use AI services to develop apps across devices and platforms. The APIs continuously improve and are easy to set up.

[] [Expand table](#)

| AI services feature | Description |
|-------------------------|--|
| Type | APIs for building intelligent applications |
| Supported languages | Various options depending on the service. The standard options are C#, Java, JavaScript, and Python. |
| Machine learning phases | Deployment |
| Key benefits | <ul style="list-style-type: none"> - Build intelligent applications by using pretrained models that are available through REST API and SDK - Use various models for natural communication methods that have vision, speech, language, and decision-making capabilities - No or minimal machine learning or data science expertise is required - The APIs are scalable and flexible - You can choose from various models |

SQL machine learning

[SQL machine learning](#) adds statistical analysis, data visualization, and predictive analytics in Python and R for relational data, both on-premises and in the cloud. Current platforms and tools include:

- [SQL Server Machine Learning Services](#).

- [SQL Managed Instance Machine Learning Services](#).
- [Machine learning in Azure Synapse Analytics](#).
- [Machine Learning extension for Azure Data Studio](#).

Use SQL machine learning when you need built-in AI and predictive analytics on relational data in SQL.

[] [Expand table](#)

| SQL machine learning feature | Description |
|-------------------------------------|--|
| Type | On-premises predictive analytics for relational data |
| Supported languages | <ul style="list-style-type: none"> - Python - R - SQL |
| Machine learning phases | <ul style="list-style-type: none"> - Data preparation - Model training - Deployment |
| Key benefits | Encapsulate predictive logic in a database function. This process makes it easy to include data-tier logic. |
| Considerations | Assumes that you use a SQL database as the data tier for your application. |

Azure AI Foundry

Azure AI Foundry is a unified platform that you can use to develop and deploy generative AI applications and Azure AI APIs responsibly. It provides a comprehensive set of AI capabilities, a simplified user interface, and code-first experiences. These features make it a comprehensive platform for building, testing, deploying, and managing intelligent solutions.

Azure AI Foundry helps developers and data scientists efficiently create and deploy generative AI applications by using Azure AI offerings. It emphasizes responsible AI development and embeds principles of fairness, transparency, and accountability. The platform includes tools for bias detection, interpretability, and privacy-preserving machine learning. These tools help ensure that AI models are powerful, trustworthy, and compliant with regulatory requirements.

As part of the Microsoft Azure ecosystem, Azure AI Foundry provides robust tools and services that cater to various AI and machine learning needs, including natural language

processing and computer vision. Its integration with other Azure services helps ensure seamless scalability and performance, which makes it an ideal option for enterprises.

The [Azure AI Foundry portal](#) fosters collaboration and innovation by providing features like shared workspaces, version control, and integrated development environments. By integrating popular open-source frameworks and tools, Azure AI Foundry accelerates the development process so that organizations can drive innovation and stay ahead in the competitive AI landscape.

[+] [Expand table](#)

| Azure AI Foundry feature | Description |
|--------------------------|---|
| Type | Unified development environment for AI |
| Supported languages | Python only |
| Machine learning phases | <ul style="list-style-type: none">- Data preparation- Deployment (Models as a service (MaaS)) |
| Key benefits | <ul style="list-style-type: none">- Facilitates collaboration and project management across various AI services- Provides comprehensive tools for building, training, and deploying AI models- Emphasizes responsible AI by providing tools for bias detection, interpretability, and privacy-preserving machine learning- Supports integration with popular open-source frameworks and tools- Includes prompt flow for creating and managing prompt-based workflows <p>Prompt flow simplifies the development cycle of AI applications that are powered by language models</p> |

Azure Machine Learning studio

[Azure Machine Learning studio](#) is a collaborative, drag-and-drop tool for building, testing, and deploying predictive analytics solutions on your data. It's designed for data scientists, data engineers, and business analysts. Machine Learning studio supports a wide range of machine learning algorithms and tools for data preparation, model training, and evaluation. It also provides a visual interface for connecting datasets and modules on an interactive canvas.

[+] [Expand table](#)

| Machine Learning studio feature | Description |
|---------------------------------|--|
| Type | Collaborative, drag-and-drop tool for machine learning |
| Supported languages | <ul style="list-style-type: none"> - Python - R - Scala - Java (limited experience) |
| Machine learning phases | <ul style="list-style-type: none"> - Data preparation - Model training - Deployment |
| Key benefits | <ul style="list-style-type: none"> - Requires no coding to build machine learning models - Supports a wide range of machine learning algorithms and tools for data preparation, model training, and evaluation - Provides a visual interface for connecting datasets and modules on an interactive canvas - Supports integration with Machine Learning for advanced machine learning tasks |

For a comprehensive comparison of Machine Learning studio and the [Azure AI Foundry portal](#), see [Azure AI Foundry portal or Machine Learning studio](#). The following table summarizes the key differences between them:

[+] Expand table

| Category | Feature | Azure AI Foundry portal | Machine Learning studio |
|------------------|---------------------|--|--|
| Data storage | Storage solution | No | Yes (cloud filesystem, OneLake, Azure Storage) |
| Data preparation | Data integration | Yes (Azure Blob Storage, OneLake, Azure Data Lake Storage) | Yes (copy and mount by using Azure storage accounts) |
| Development | Code-first tools | Yes (VS Code) | Yes (Notebooks, Jupyter, VS Code, R Studio) |
| Languages | Supported languages | Python only | Python, R, Scala, Java |
| Training | AutoML | No | Yes (regression, classification, forecasting, CV, NLP) |
| Compute targets | Training compute | Serverless (MaaS, prompt flow) | Spark clusters, machine learning clusters, Azure Arc |

| Category | Feature | Azure AI Foundry portal | Machine Learning studio |
|---------------|-----------------------------|--|--|
| Generative AI | Language model catalog | Yes (Azure OpenAI, Hugging Face, Meta) | Yes (Azure OpenAI, Hugging Face, Meta) |
| Deployment | Real-time and batch serving | Real-time (MaaS) | Batch endpoints, Azure Arc |
| Governance | Responsible AI tools | No | Yes (Responsible AI dashboard) |

Microsoft Fabric

Fabric is an end-to-end, unified analytics platform that brings together all the data and analytics tools that organizations need. It integrates various services and tools to provide a seamless experience for data professionals, including data engineers, data scientists, and business analysts. Fabric provides capabilities for data integration, data engineering, data warehousing, data science, real-time analytics, and business intelligence.

Use Fabric when you need a comprehensive platform to manage your entire data lifecycle from ingestion to insights.

[\[+\] Expand table](#)

| Fabric feature | Description |
|-------------------------|---|
| Type | Unified analytics platform |
| Supported languages | <ul style="list-style-type: none"> - Python - R - SQL - Scala |
| Machine learning phases | <ul style="list-style-type: none"> - Data preparation - Model training - Deployment - Real-time analytics |
| Key benefits | <ul style="list-style-type: none"> - Unified platform for all data and analytics needs - Seamless integration with other Microsoft services - Scalable and flexible - Supports a wide range of data and analytics tools - Facilitates collaboration across different roles in an organization - End-to-end data lifecycle management from ingestion to insights - Real-time analytics and business intelligence capabilities - Machine learning model training and deployment support |

| Fabric feature | Description |
|----------------|--|
| | <ul style="list-style-type: none"> - Integration with popular machine learning frameworks and tools - Tools for data preparation and feature engineering - Real-time machine learning inference and analytics |

Azure Data Science Virtual Machine

[Azure Data Science Virtual Machine](#) is a customized VM environment on the Microsoft Azure cloud. It's available in versions for both Windows and Linux Ubuntu. The environment is specifically for data science tasks and machine learning solution development. It has many popular data science functions, machine learning frameworks, and other tools that are preinstalled and preconfigured so that you can jump-start building intelligent applications for advanced analytics.

Use the Data Science VM when you need to run or host your jobs on a single node or if you need to remotely scale up your processing on a single machine.

[+] [Expand table](#)

| Azure Data Science Virtual Machine feature | Description |
|--|--|
| Type | Customized VM environment for data science |
| Key benefits | <ul style="list-style-type: none"> - Reduced time to install, manage, and troubleshoot data science tools and frameworks - Includes the latest versions of commonly used tools and frameworks - Includes highly scalable images and graphics processing unit (GPU) capabilities for intensive data modeling |
| Considerations | <ul style="list-style-type: none"> - The VM can't be accessed when it's offline. - Running a VM incurs Azure charges, so you should make sure that it runs only when you need it. |

Azure Databricks

[Azure Databricks](#) is an Apache Spark-based analytics platform that's optimized for the Microsoft Azure cloud platform. Azure Databricks is integrated with Azure to provide one-click setup, streamlined workflows, and an interactive workspace that enables collaboration between data scientists, data engineers, and business analysts. Use Python, R, Scala, and SQL code in web-based notebooks to query, visualize, and model data.

Use Azure Databricks when you want to collaborate on building machine learning solutions on Apache Spark.

[+] Expand table

| Azure Databricks feature | Description |
|--------------------------|---|
| Type | Apache Spark-based analytics platform |
| Supported languages | <ul style="list-style-type: none">- Python- R- Scala- SQL |
| Machine learning phases | <ul style="list-style-type: none">- Data preparation- Data preprocessing- Model training- Model tuning- Model inference- Management- Deployment |
| Key benefits | <ul style="list-style-type: none">- One-click setup and streamlined workflows for easy use- Interactive workspace for collaboration- Seamless integration with Azure- Scalability to handle large datasets and intensive computations- Support for various languages and integration with popular tools |

ML.NET

[ML.NET](#) is an open-source, cross-platform machine learning framework. Use ML.NET to build custom machine learning solutions and integrate them into your .NET applications. ML.NET provides various levels of interoperability with popular frameworks like TensorFlow and ONNX for training and scoring machine learning and deep learning models. For resource-intensive tasks like training image classification models, you can use Azure to train your models in the cloud.

Use ML.NET when you want to integrate machine learning solutions into your .NET applications. Choose between the [API](#) for a code-first experience and [Model Builder](#) or the [CLI](#) for a low-code experience.

[+] Expand table

| ML.NET feature | Description |
|-------------------------|---|
| Type | Open-source, cross-platform framework for developing custom machine learning applications with .NET |
| Supported languages | - C# - F# |
| Machine learning phases | - Data preparation - Training - Deployment |
| Key benefits | - No requirement for data science or machine learning experience - Familiar languages and tools like Visual Studio and VS Code - Deploys the application where .NET runs - Extensible and scalable design - Local-first experience - AutoML for automated machine learning tasks |

AI for Windows apps

Use [AI for Windows apps](#) to integrate AI capabilities into Windows applications. Use WinML and DirectML capabilities to provide local, real-time AI model evaluation and hardware acceleration. WinML allows developers to integrate trained machine learning models directly into their Windows applications. It facilitates local, real-time evaluation of models and enables powerful AI capabilities without the need for cloud connectivity.

DirectML is a high-performance, hardware-accelerated platform for running machine learning models. It uses DirectX APIs to provide optimized performance across diverse hardware, including GPUs and AI accelerators.

Use AI for Windows apps when you want to use trained machine learning models within your Windows applications.

[Expand table](#)

| AI for Windows apps feature | Description |
|------------------------------------|--|
| Type | Inference engine for trained models in Windows devices |
| Supported languages | - C#/C++ - JavaScript |
| Machine learning phases | - Data preparation - Model training |

| AI for Windows apps feature | Description |
|-----------------------------|--|
| | - Deployment |
| Key benefits | - Local, real-time AI model evaluation - High-performance AI processing across various hardware types, including CPUs, GPUs, and AI accelerators - Consistent behavior and performance across Windows hardware |

SynapseML

[SynapseML](#), formerly known as MMLSpark, is an open-source library that simplifies the creation of massively scalable machine learning pipelines. SynapseML provides APIs for various machine learning tasks, such as text analytics, vision, and anomaly detection. SynapseML is built on the [Apache Spark](#) distributed computing framework and shares the same API as the SparkML and MLlib libraries, so you can seamlessly embed SynapseML models into existing Apache Spark workflows.

SynapseML adds many deep learning and data science tools to the Spark ecosystem, including seamless integration of [Spark Machine Learning](#) pipelines with [Light Gradient Boosting Machine \(LightGBM\)](#), [Local Interpretable Model-Agnostic Explanations](#), and [OpenCV](#). You can use these tools to create powerful predictive models on any Spark cluster, such as [Azure Databricks](#) or [Azure Cosmos DB](#).

SynapseML also provides networking capabilities to the Spark ecosystem. With the HTTP on Spark project, users can embed any web service into their SparkML models. Additionally, SynapseML provides easy-to-use tools for orchestrating [AI services](#) at scale. For production-grade deployment, the Spark Serving project enables high throughput and submillisecond latency web services that are backed by your Spark cluster.

[\[+\] Expand table](#)

| SynapseML feature | Description |
|---------------------|--|
| Type | Open-source, distributed machine learning and microservices framework for Apache Spark |
| Supported languages | - Scala - Java - Python - R - .NET |

| SynapseML feature | Description |
|-------------------------|---|
| Machine learning phases | <ul style="list-style-type: none"> - Data preparation - Model training - Deployment |
| Key benefits | <ul style="list-style-type: none"> - Scalability - Streaming and serving compatible - High fault tolerance |
| Considerations | Requires Apache Spark |

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [Mahdi Setayesh](#) | Principal Software Engineer

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- [AI development products available from Microsoft](#)
- [Microsoft Learn training in developing AI and machine learning solutions](#)
- [How Azure Machine Learning works](#)

Related resources

- [Choose an Azure AI services technology](#)
- [AI architecture design](#)

Feedback

Was this page helpful?



AI agent orchestration patterns

07/18/2025

As architects and developers design their workload to take full advantage of language model capabilities, AI agent systems become increasingly complex. These systems often exceed the abilities of a single agent that has access to many tools and knowledge sources. Instead, these systems use multi-agent orchestrations to handle complex, collaborative tasks reliably. This guide covers fundamental orchestration patterns for multi-agent architectures and helps you choose the approach that fits your specific requirements.

Overview

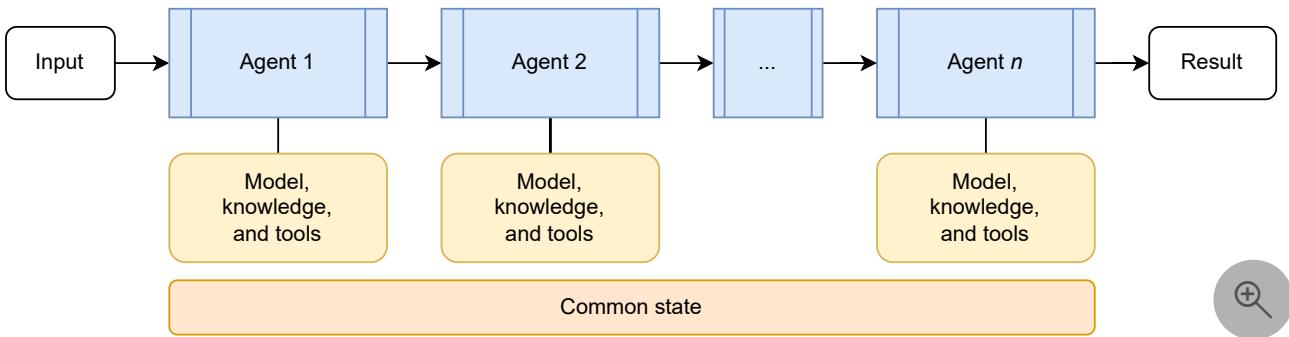
When you use multiple AI agents, you can break down complex problems into specialized units of work or knowledge. You assign each task to dedicated AI agents that have specific capabilities. These approaches mirror strategies found in human teamwork. Using multiple agents provides several advantages compared to monolithic single-agent solutions.

- **Specialization:** Individual agents can focus on a specific domain or capability, which reduces code and prompt complexity.
- **Scalability:** Agents can be added or modified without redesigning the entire system.
- **Maintainability:** Testing and debugging can be focused on individual agents, which reduces the complexity of these tasks.
- **Optimization:** Each agent can use distinct models, task-solving approaches, knowledge, tools, and compute to achieve its outcomes.

The patterns in this guide show proven approaches for orchestrating multiple agents to work together and accomplish an outcome. Each pattern is optimized for different types of coordination requirements. These AI agent orchestration patterns complement and extend traditional [cloud design patterns](#) by addressing the unique challenges of coordinating autonomous components in AI-driven workload capabilities.

Sequential orchestration

The sequential orchestration pattern chains AI agents in a predefined, linear order. Each agent processes the output from the previous agent in the sequence, which creates a pipeline of specialized transformations.



The sequential orchestration pattern solves problems that require step-by-step processing, where each stage builds on the previous stage. It suits workflows that have clear dependencies and improve output quality through progressive refinement. This pattern resembles the [Pipes and Filters](#) cloud design pattern, but it uses AI agents instead of custom-coded processing components. The choice of which agent gets invoked next is deterministically defined as part of the workflow and isn't a choice given to agents in the process.

When to use sequential orchestration

Consider the sequential orchestration pattern in the following scenarios:

- Multistage processes that have clear linear dependencies and predictable workflow progression
- Data transformation pipelines, where each stage adds specific value that the next stage depends on
- Workflow stages that can't be parallelized
- Progressive refinement requirements, such as *draft*, *review*, *polish* workflows
- Systems where you understand the availability and performance characteristics of every AI agent in the pipeline, and where failures or delays in one AI agent's processing are tolerable for the overall task to be accomplished

When to avoid sequential orchestration

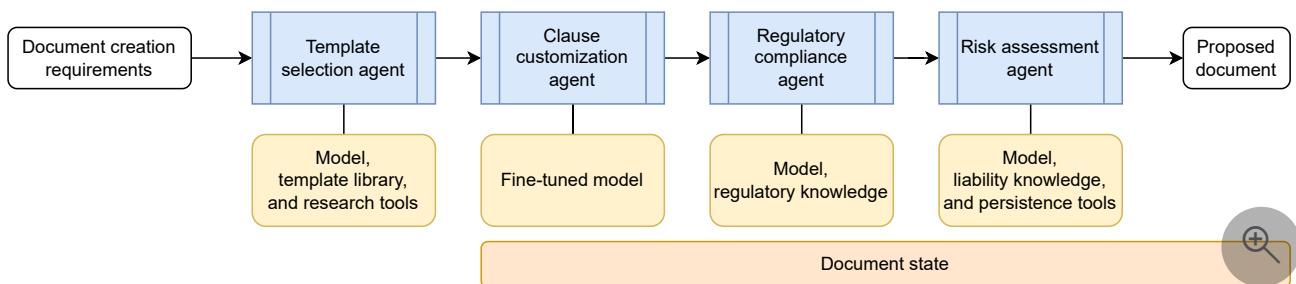
Avoid this pattern in the following scenarios:

- Stages are [embarrassingly parallel](#). You can parallelize them without compromising quality or creating shared state contention.
- Processes that include only a few stages that a single AI agent can accomplish effectively.

- Early stages might fail or produce low-quality output, and there's no reasonable way to prevent later steps from processing by using accumulated error output.
- AI agents need to collaborate rather than hand off work.
- The workflow requires backtracking or iteration.
- You need dynamic routing based on intermediate results.

Sequential orchestration example

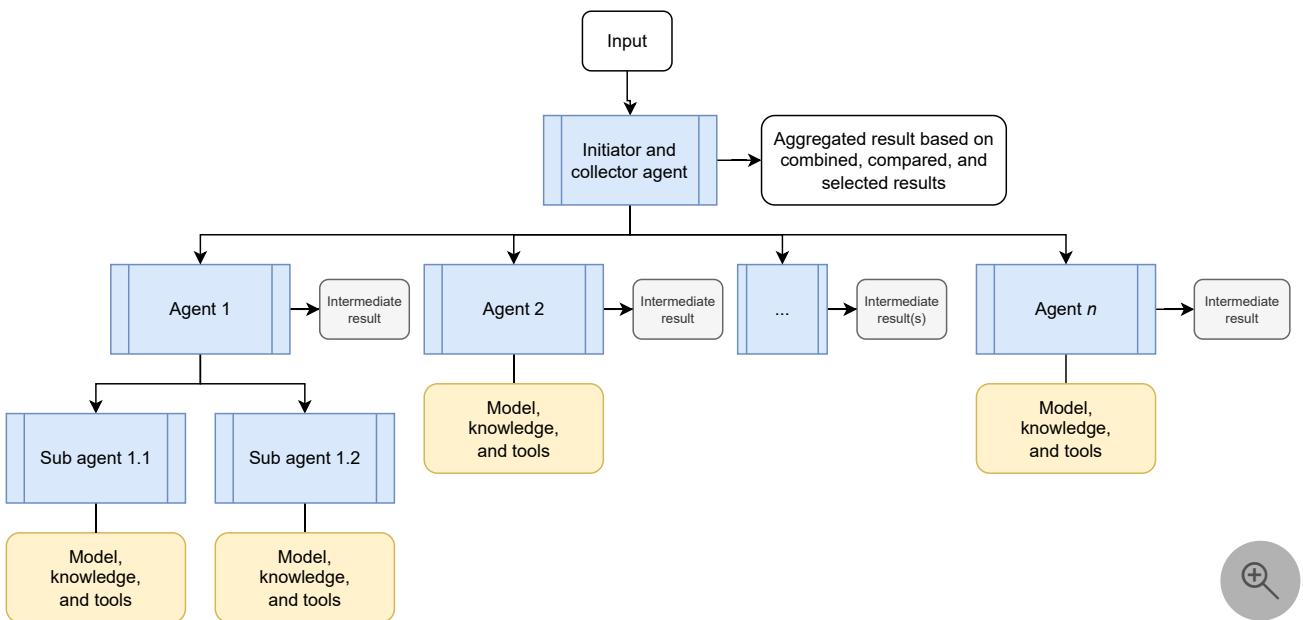
A law firm's document management software uses sequential agents for contract generation. The intelligent application processes requests through a pipeline of four specialized agents. The sequential and predefined pipeline steps ensure that each agent works with the complete output from the previous stage.



1. The *template selection agent* receives client specifications, like contract type, jurisdiction, and parties involved, and selects the appropriate base template from the firm's library.
2. The *clause customization agent* takes the selected template and modifies standard clauses based on negotiated business terms, including payment schedules and liability limitations.
3. The *regulatory compliance agent* reviews the customized contract against applicable laws and industry-specific regulations.
4. The *risk assessment agent* performs comprehensive analysis of the complete contract. It evaluates liability exposure and dispute resolution mechanisms while providing risk ratings and protective language recommendations.

Concurrent orchestration

The concurrent orchestration pattern runs multiple AI agents simultaneously on the same task. This approach allows each agent to provide independent analysis or processing from its unique perspective or specialization.



This pattern addresses scenarios where you need diverse insights or approaches to the same problem. Instead of sequential processing, all agents work in parallel, which reduces overall run time and provides comprehensive coverage of the problem space. This orchestration pattern resembles the Fan-out/Fan-in cloud design pattern. The results from each agent are often aggregated to return a final result, but that's not required. Each agent can independently produce its own results within the workload, such as invoking tools to accomplish tasks or updating different data stores in parallel.

Agents operate independently and don't hand off results to each other. An agent might invoke extra AI agents by using its own orchestration approach as part of its independent processing. The available agents must know which agents are available for processing. This pattern supports both deterministic calls to all registered agents and dynamic selection of which agents to invoke based on the task requirements.

When to use concurrent orchestration

Consider the concurrent orchestration pattern in the following scenarios:

- Tasks that you can run in parallel, either by using a fixed set of agents or by dynamically choosing AI agents based on specific task requirements.
- Tasks that benefit from multiple independent perspectives or different specializations, such as technical, business, and creative approaches, that can all contribute to the same problem. This collaboration typically occurs in scenarios that feature the following multi-agent decision-making techniques:
 - Brainstorming

- Ensemble reasoning
- Quorum and voting-based decisions
- Time-sensitive scenarios where parallel processing reduces latency.

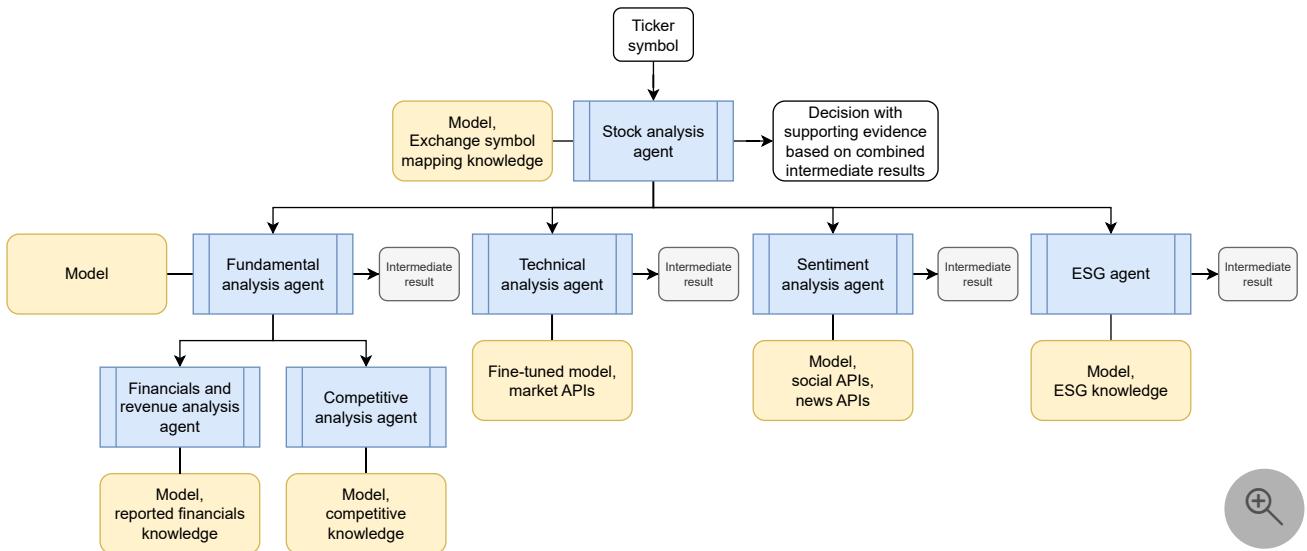
When to avoid concurrent orchestration

Avoid this orchestration pattern in the following scenarios:

- Agents need to build on each other's work or require cumulative context in a specific sequence.
- The task requires a specific order of operations or deterministic, reproducible results from running in a defined sequence.
- Resource constraints, such as model quota, make parallel processing inefficient or impossible.
- Agents can't reliably coordinate changes to shared state or external systems while running simultaneously.
- There's no clear conflict resolution strategy to handle contradictory or conflicting results from each agent.
- Result aggregation logic is too complex or lowers the quality of the results.

Concurrent orchestration example

A financial services firm built an intelligent application that uses concurrent agents that specialize in different types of analysis to evaluate the same stock simultaneously. Each agent contributes insights from its specialized perspective, which provides diverse, time-sensitive input for rapid investment decisions.



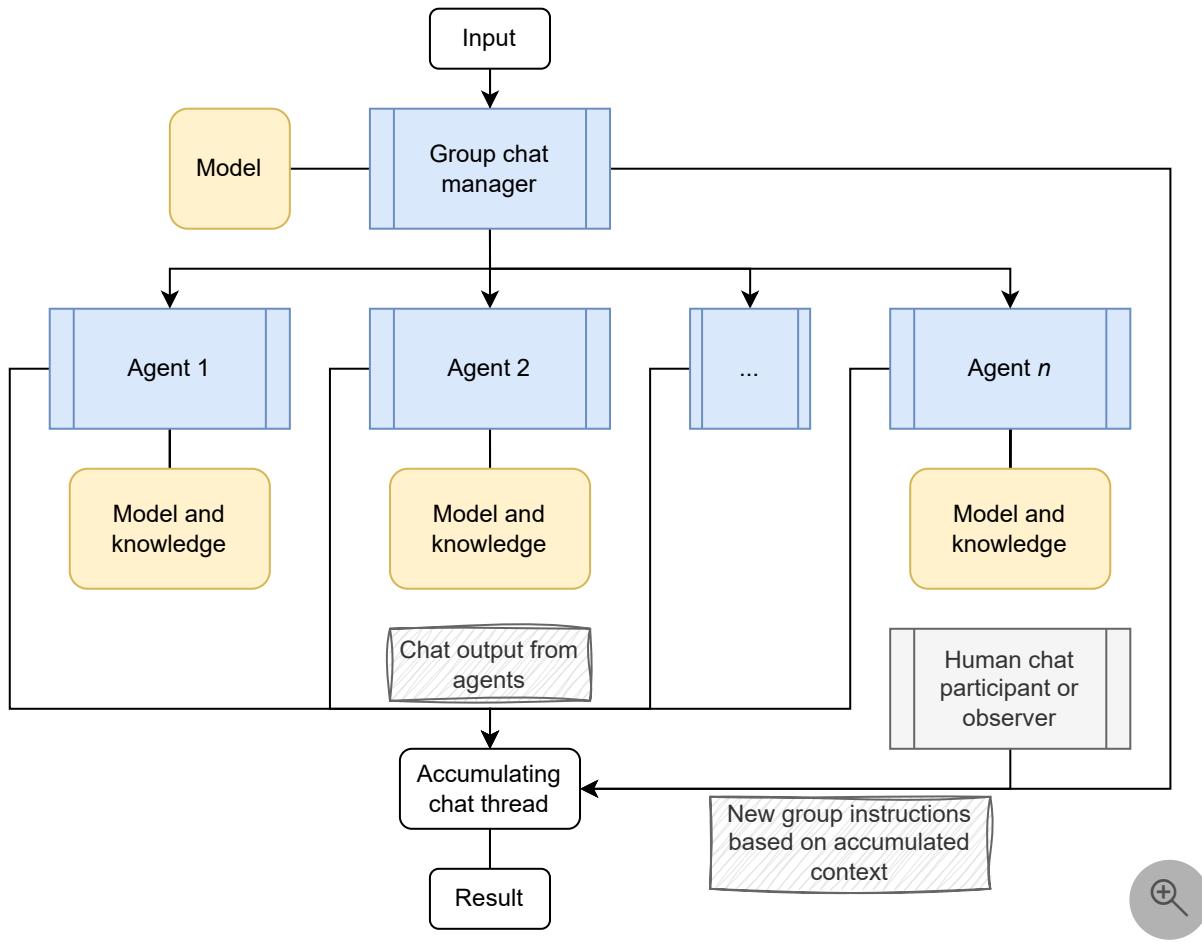
The system processes stock analysis requests by dispatching the same ticker symbol to four specialized agents that run in parallel.

- The *fundamental analysis agent* evaluates financial statements, revenue trends, and competitive positioning to assess intrinsic value.
- The *technical analysis agent* examines price patterns, volume indicators, and momentum signals to identify trading opportunities.
- The *sentiment analysis agent* processes news articles, social media mentions, and analyst reports to gauge market sentiment and investor confidence.
- The *environmental, social, and governance (ESG) agent* reviews environmental impact, social responsibility, and governance practice reports to evaluate sustainability risks and opportunities.

These independent results are then combined into a comprehensive investment recommendation, which enables portfolio managers to make informed decisions quickly.

Group chat orchestration

The group chat orchestration pattern enables multiple agents to solve problems, make decisions, or validate work by participating in a shared conversation thread where they collaborate through discussion. A chat manager coordinates the flow by determining which agents can respond next and by managing different interaction modes, from collaborative brainstorming to structured quality gates.



This pattern addresses scenarios that are best accomplished through group discussion to reach decisions. These scenarios might include collaborative ideation, structured validation, or quality control processes. The pattern supports various interaction modes, from free-flowing brainstorming to formal review workflows that have fixed roles and approval gates.

This pattern works well for human-in-the-loop scenarios where humans can optionally take on dynamic chat manager responsibilities and guide conversations toward productive outcomes. In this orchestration pattern, agents are typically in a *read-only* mode. They don't use tools to make changes in running systems.

When to use group chat orchestration

Consider group chat orchestration when your scenario can be solved through spontaneous or guided collaboration or iterative maker-checker loops. All of these approaches support real-time human oversight or participation. Because all agents and humans in the loop emit output into a single accumulating thread, this pattern provides transparency and auditability.

Collaborative scenarios

- Creative brainstorming sessions where agents that have different perspectives and knowledge sources build on each other's contributions to the chat
- Decision-making processes that benefit from debate and consensus-building
- Decision-making scenarios that require iterative refinement through discussion
- Multidisciplinary problems that require cross-functional dialogue

Validation and quality control scenarios

- Quality assurance requirements that involve structured review processes and iteration
- Compliance and regulatory validation that requires multiple expert perspectives
- Content creation workflows that require editorial review with a clear separation of concerns between creation and validation

When to avoid group chat orchestration

Avoid this pattern in the following scenarios:

- Simple task delegation or linear pipeline processing is sufficient.
- Real-time processing requirements make discussion overhead unacceptable.
- Clear hierarchical decision-making or deterministic workflows without discussion are more appropriate.
- The chat manager has no objective way to determine whether the task is complete.

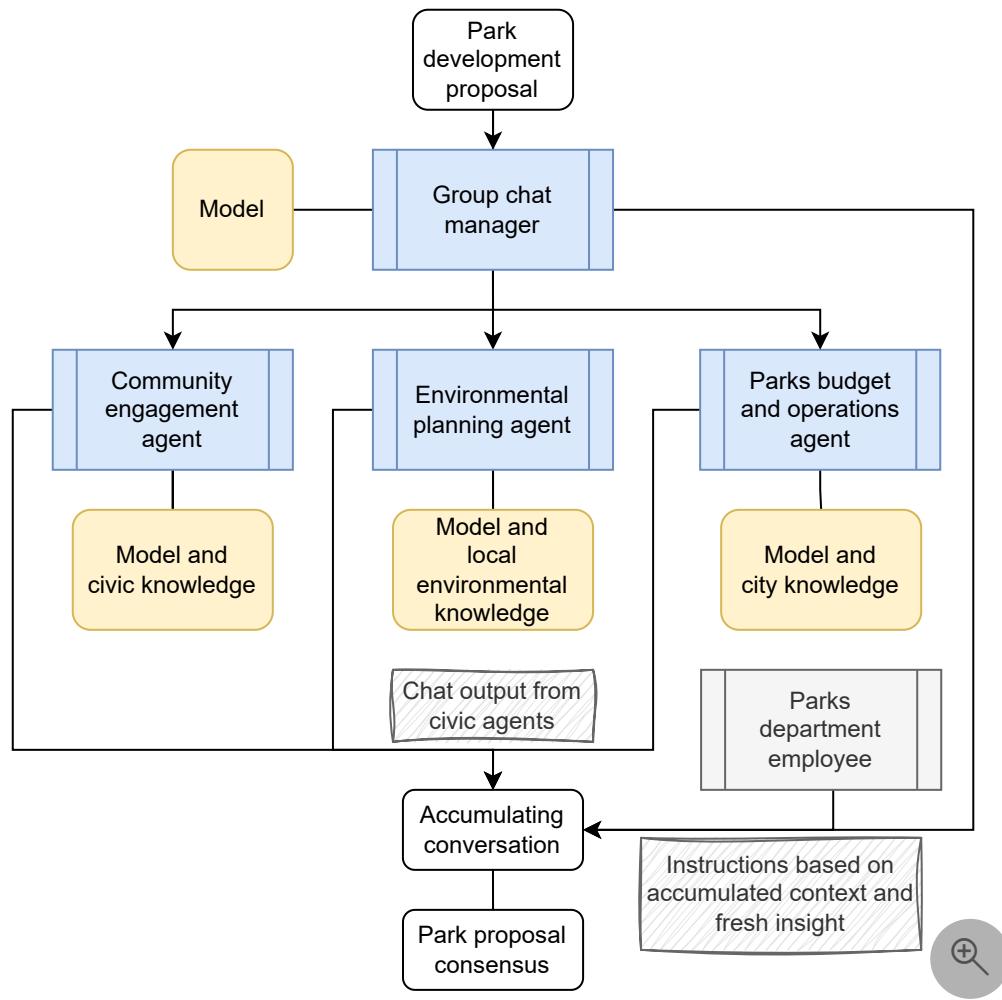
Managing conversation flow and preventing infinite loops require careful attention, especially as more agents make control more difficult to maintain. To maintain effective control, consider limiting group chat orchestration to three or fewer agents.

Maker-checker loops

The maker-checker loop is a specific type of group chat orchestration where one agent, the *maker*, creates or proposes something. Another agent, the *checker*, provides a critique of the result. This pattern is iterative, with the checker agent pushing the conversation back to the maker agent to make updates and repeat the process. Although the group chat pattern doesn't require agents to *take turns* chatting, the maker-checker loop requires a formal turn-based sequence that the chat manager drives.

Group chat orchestration example

A city parks and recreation department uses software that includes group chat orchestration to evaluate new park development proposals. The software reads the draft proposal, and multiple specialist agents debate different community impact perspectives and work toward consensus on the proposal. This process occurs before the proposal opens for community review to help anticipate the feedback that it might receive.



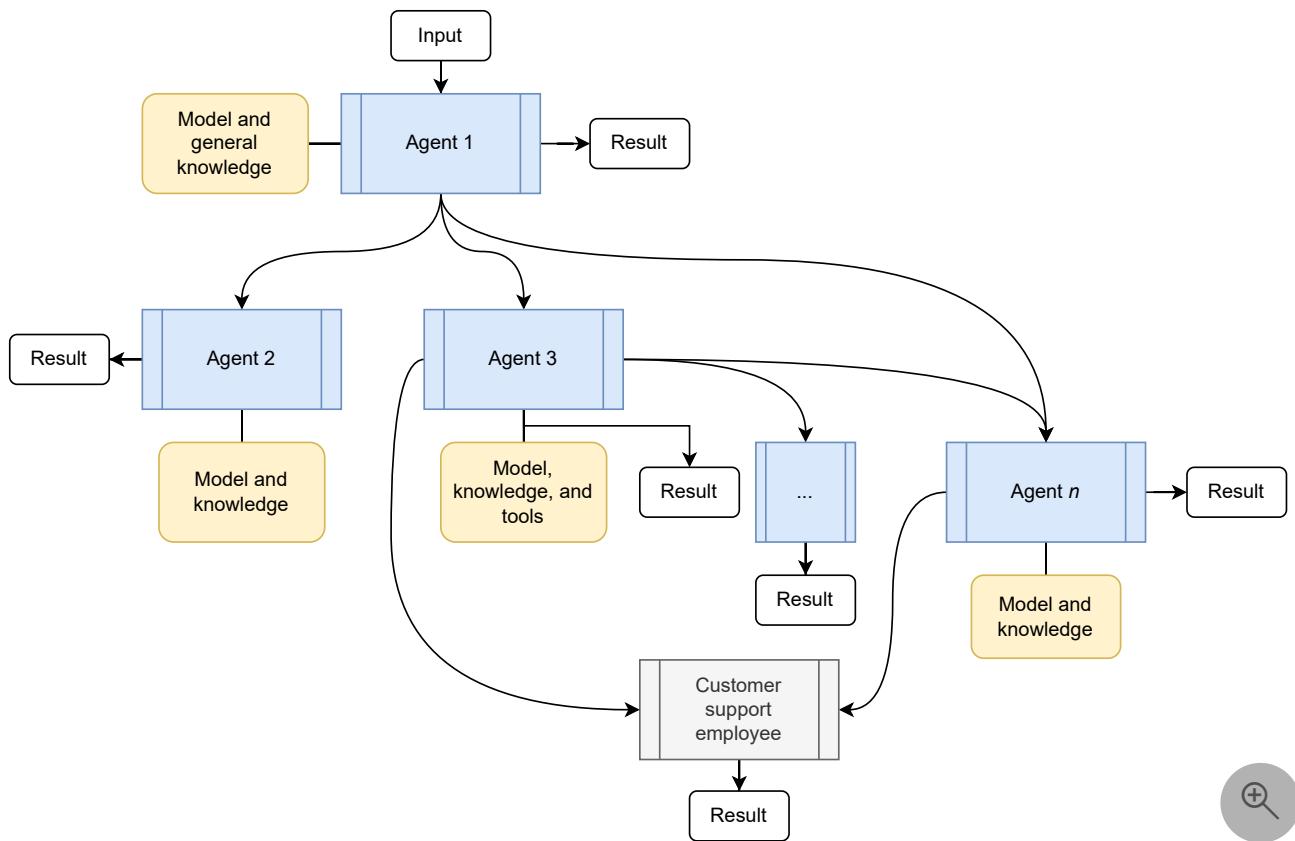
The system processes park development proposals by initiating a group consultation with specialized municipal agents that engage in the task from multiple civic perspectives.

- The *community engagement agent* evaluates accessibility requirements, anticipated resident feedback, and usage patterns to ensure equitable community access.
- The *environmental planning agent* assesses ecological impact, sustainability measures, native vegetation displacement, and compliance with environmental regulations.
- The *budget and operations agent* analyzes construction costs, ongoing maintenance expenses, staffing requirements, and long-term operational sustainability.

The chat manager facilitates structured debate where agents challenge each other's recommendations and defend their reasoning. A parks department employee participates in the chat thread to add insight and respond to agents' knowledge requests in real time. This process enables the employee to update the original proposal to address identified concerns and better prepare for community feedback.

Handoff orchestration

The handoff orchestration pattern enables dynamic delegation of tasks between specialized agents. Each agent can assess the task at hand and decide whether to handle it directly or transfer it to a more appropriate agent based on the context and requirements.



This pattern addresses scenarios where the optimal agent for a task isn't known upfront or where the task requirements become clear only during processing. It enables intelligent routing and ensures that tasks reach the most capable agent. Agents in this pattern don't typically work in parallel. Full control transfers from one agent to another agent.

When to use handoff orchestration

Consider the agent handoff pattern in the following scenarios:

- Tasks that require specialized knowledge or tools, but where the number of agents needed or their order can't be predetermined
- Scenarios where expertise requirements emerge during processing, resulting in dynamic task routing based on content analysis
- Multiple-domain problems that require different specialists who operate one at a time
- Logical relationships and signals that you can predetermine to indicate when one agent reaches its capability limit and which agent should handle the task next

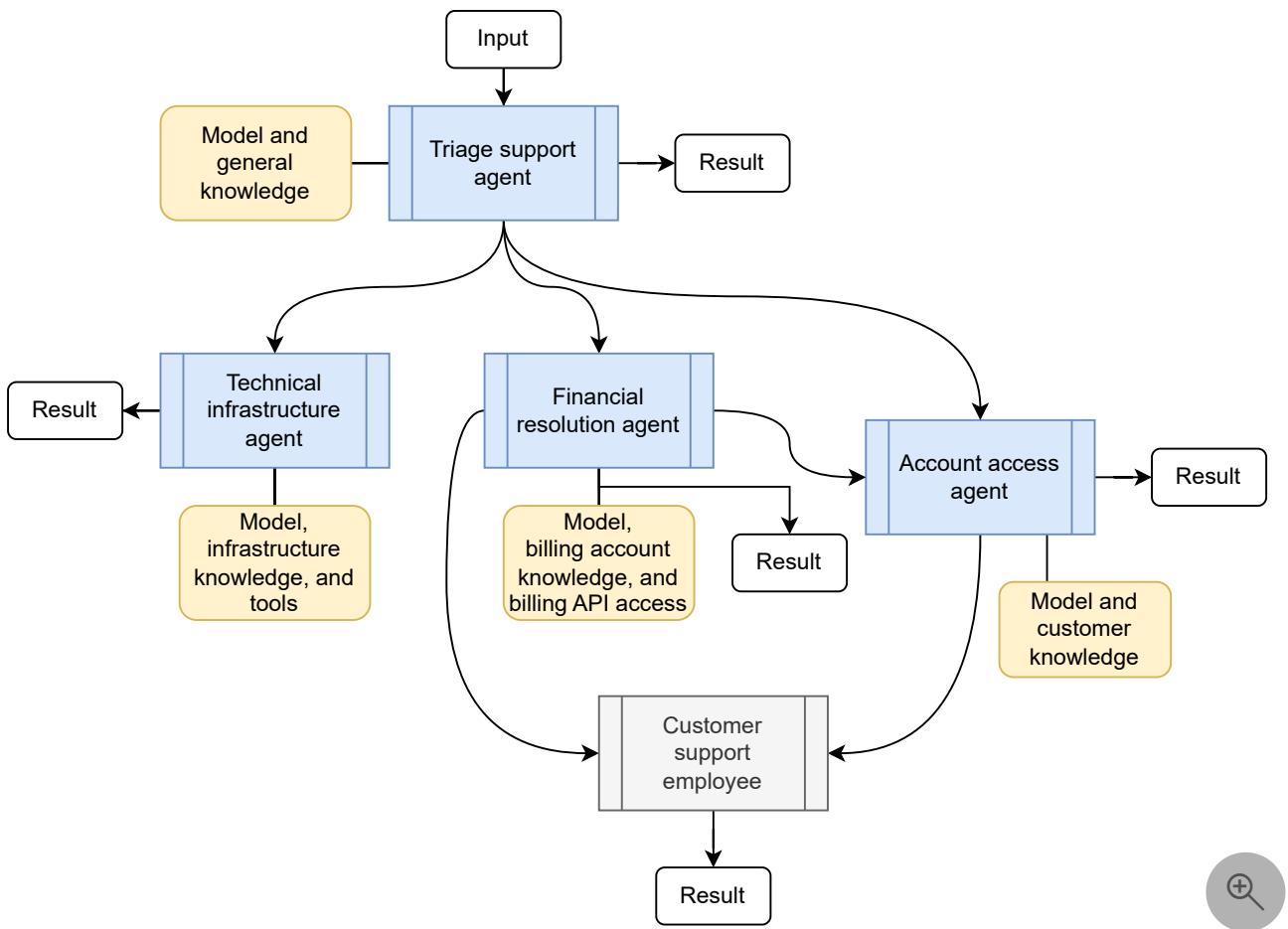
When to avoid handoff orchestration

Avoid this pattern in the following scenarios:

- The appropriate agents and their order are always known upfront.
- Task routing is simple and deterministically rule-based, not based on dynamic context window or dynamic interpretation.
- Suboptimal routing decisions might lead to a poor or frustrating user experience.
- Multiple operations should run concurrently to address the task.
- Avoiding an infinite handoff loop or avoiding excessive bouncing between agents is challenging.

Agent handoff pattern example

A telecommunications customer relationship management (CRM) solution uses handoff agents in its customer support web portal. An initial agent begins helping customers but discovers that it needs specialized expertise during the conversation. The initial agent passes the task to the most appropriate agent to address the customer's concern. Only one agent at a time operates on the original input, and the handoff chain results in a single result.



In this system, the *triage support agent* interprets the request and tries to handle common problems directly. When it reaches its limits, it hands network problems to a *technical infrastructure agent*, billing disputes to a *financial resolution agent*, and so on. Further handoffs occur within those agents when the current agent recognizes its own capability limits and knows another agent can better support the scenario.

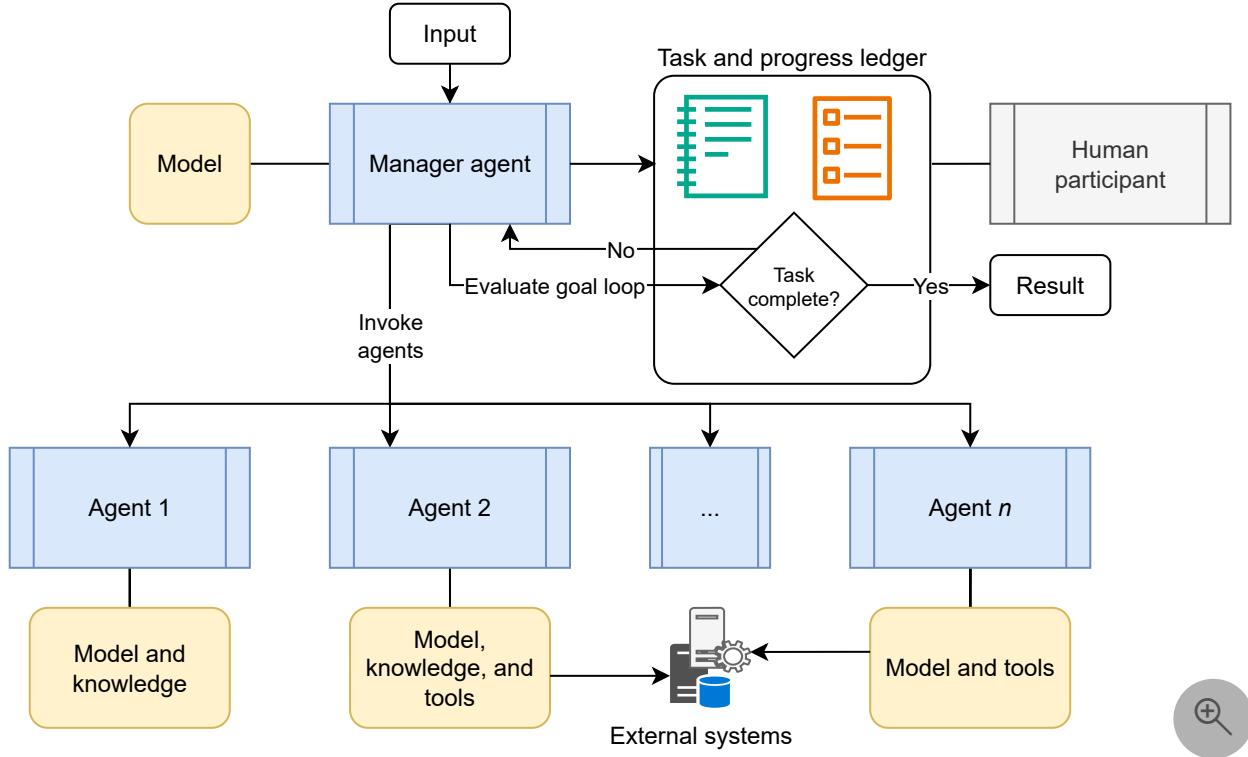
Each agent is capable of completing the conversation if it determines that customer success has been achieved or that no other agent can further benefit the customer. Some agents are also designed to hand off the user experience to a human support agent when the problem is important to solve but no AI agent currently has the capabilities to address it.

One example of a handoff instance is highlighted in the diagram. It begins with the triage agent that hands off the task to the technical infrastructure agent. The technical infrastructure agent then decides to hand off the task to the financial resolution agent, which ultimately redirects the task to customer support.

Magnetic orchestration

The magnetic orchestration pattern is designed for open-ended and complex problems that don't have a predetermined plan of approach. Agents in this pattern typically have tools that allow them to make direct changes in external systems. The focus is as much on building and

documenting the approach to solve the problem as it is on implementing that approach. The task list is dynamically built and refined as part of the workflow through collaboration between specialized agents and a magentic manager agent. As the context evolves, the magentic manager agent builds a task ledger to develop the approach plan with goals and subgoals, which is eventually finalized, followed, and tracked to complete the desired outcome.



The manager agent communicates directly with specialized agents to gather information as it builds and refines the task ledger. It iterates, backtracks, and delegates as many times as needed to build a complete plan that it can successfully carry out. The manager agent frequently evaluates whether the original request is fully satisfied or stalled. It updates the ledger to adjust the plan.

In some ways, this orchestration pattern is an extension of the [group chat](#) pattern. The magentic orchestration pattern focuses on an agent that builds a plan of approach, while other agents use tools to make changes in external systems instead of only using their knowledge stores to reach an outcome.

When to use magentic orchestration

Consider the magentic pattern in the following scenarios:

- A complex or open-ended use case that has no predetermined solution path.
- A requirement to consider input and feedback from multiple specialized agents to develop a valid solution path.

- A requirement for the AI system to generate a fully developed plan of approach that a human can review before or after implementation.
- Agents equipped with tools that interact with external systems, consume external resources, or can induce changes in running systems. A documented plan that shows how those agents are sequenced can be presented to a user before allowing the agents to follow the tasks.

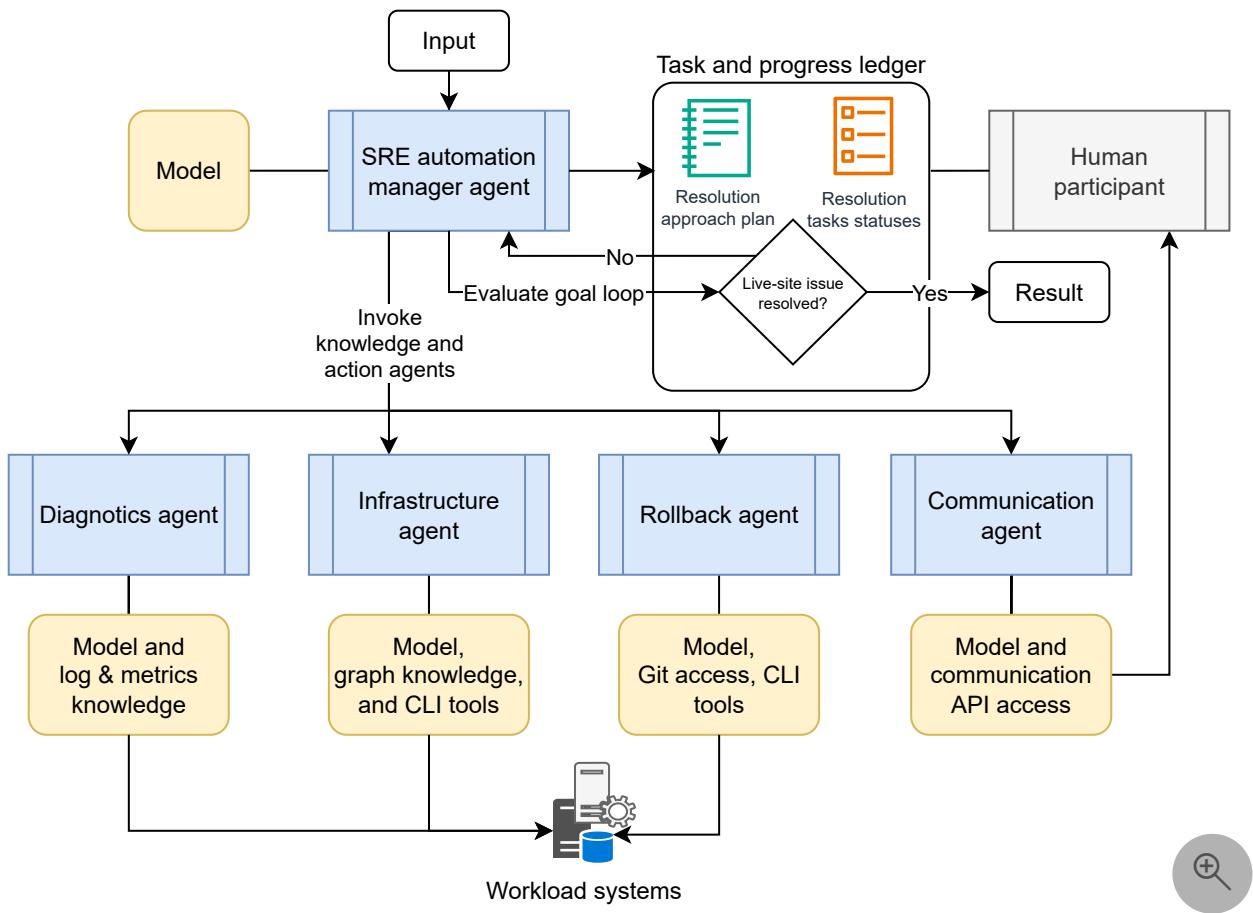
When to avoid magentic orchestration

Avoid this pattern in the following scenarios:

- The solution path is developed or should be approached in a deterministic way.
- There's no requirement to produce a ledger.
- The task has low complexity and a simpler pattern can solve it.
- The work is time-sensitive, as the pattern focuses on building and debating viable plans, not optimizing for end results.
- You anticipate frequent stalls or infinite loops that don't have a clear path to resolution.

Magnetic orchestration example

A site reliability engineering (SRE) team built automation that uses magnetic orchestration to handle low-risk incident response scenarios. When a service outage occurs within the scope of the automation, the system must dynamically create and implement a remediation plan. It does this without knowing the specific steps needed upfront.



When the automation detects a qualifying incident, the *magnetic manager agent* begins by creating an initial task ledger with high-level goals such as restoring service availability and identifying the root cause. The manager agent then consults with specialized agents to gather information and refine the remediation plan.

1. The *diagnostics agent* analyzes system logs, performance metrics, and error patterns to identify potential causes. It reports findings back to the manager agent.
2. Based on diagnostic results, the manager agent updates the task ledger with specific investigation steps and consults the *infrastructure agent* to understand current system state and available recovery options.
3. The *communication agent* provides stakeholder notification capabilities, and the manager agent incorporates communication checkpoints and approval gates into the evolving plan according to the SRE team's escalation procedures.
4. As the scenario becomes clearer, the manager agent might add the *rollback agent* to the plan if deployment reversion is needed, or escalate to human SRE engineers if the incident exceeds the automation's scope.

Throughout this process, the manager agent continuously refines the task ledger based on new information. It adds, removes, or reorders tasks as the incident evolves. For example, if the

diagnostics agent discovers a database connection problem, the manager agent might switch the entire plan from a deployment rollback strategy to a plan that focuses on restoring database connectivity.

The manager agent watches for excessive stalls in restoring service and guards against infinite remediation loops. It maintains a complete audit trail of the evolving plan and the implementation steps, which provides transparency for post-incident review. This transparency ensures that the SRE team can improve both the workload and the automation based on lessons learned.

Implementation considerations

When you implement any of these agent design patterns, several considerations must be addressed. Reviewing these considerations helps you avoid common pitfalls and ensures that your agent orchestration is robust, secure, and maintainable.

Single agent, multitool

You can address some problems with a single agent if you give it sufficient access to tools and knowledge sources. As the number of knowledge sources and tools increases, it becomes difficult to provide a predictable agent experience. If a single agent can reliably solve your scenario, consider adopting that approach. Decision-making and flow-control overhead often exceed the benefits of breaking the task into multiple agents. However, security boundaries, network line of sight, and other factors can still render a single-agent approach infeasible.

Deterministic routing

Some patterns require you to route flow between agents deterministically. Others rely on agents to choose their own routes. If your agents are defined in a no-code or low-code environment, you might not control those behaviors. If you define your agents in code by using SDKs like Semantic Kernel, you have more control.

Context window

All agents often have limited context windows. This constraint can affect their ability to process complex tasks. When you implement these patterns, decide what context the next agent requires to be effective. In some scenarios, you need the full, raw context gathered so far. In other scenarios, a summarized or truncated version is more appropriate. If your agent can work without accumulated context and only requires a new instruction set, take that approach instead of providing context that doesn't help accomplish the agent's task.

Reliability

These patterns require properly functioning agents and reliable transitions between them. They often result in classical distributed systems problems such as node failures, network partitions, message loss, and cascading errors. Mitigation strategies should be in place to address these challenges. Agents and their orchestrators should do the following steps.

- Implement timeout and retry mechanisms.
- Include a graceful degradation implementation to handle one or more agents within a pattern faulting.
- Surface errors instead of hiding them, so downstream agents and orchestrator logic can respond appropriately.
- Consider circuit breaker patterns for agent dependencies.
- Design agents to be as isolated as is practical from each other, with single points of failure not shared between agents. For example:
 - Ensure compute isolation between agents.
 - Evaluate how using a single models as a service (MaaS) model or a shared knowledge store can result in rate limiting when agents run concurrently.
- Use checkpoint features available in your SDK to help recover from an interrupted orchestration, such as from a fault or a new code deployment.

Security

Implementing proper security mechanisms in these design patterns minimizes the risk of exposing your AI system to attacks or data leakage. Securing communication between agents and limiting each agent's access to sensitive data are key security design strategies. Consider the following security measures:

- Implement authentication and use secure networking between agents.
- Consider data privacy implications of agent communications.
- Design audit trails to meet compliance requirements.
- Design agents and their orchestrators to follow the principle of least privilege.
- Consider how to handle the user's identity across agents. Agents must have broad access to knowledge stores to handle requests from all users, but they must not return data

that's inaccessible to the user. Security trimming must be implemented in every agent in the pattern.

Observability and testing

Distributing your AI system across multiple agents requires monitoring and testing each agent individually, as well as the system as a whole, to ensure proper functionality. When you design your observability and testing strategies, consider the following recommendations:

- Instrument all agent operations and handoffs. Troubleshooting distributed systems is a computer science challenge, and orchestrated AI agents are no exception.
- Track performance and resource usage metrics for each agent so that you can establish a baseline, find bottlenecks, and optimize.
- Design testable interfaces for individual agents.
- Implement integration tests for multi-agent workflows.

Common pitfalls and anti-patterns

Avoid these common mistakes when you implement agent orchestration patterns:

- Creating unnecessary coordination complexity by using a complex pattern when simple sequential or concurrent orchestration would suffice.
- Adding agents that don't provide meaningful specialization.
- Overlooking latency impacts of multiple-hop communication.
- Sharing mutable state between concurrent agents, which can result in transactionally inconsistent data because of assuming synchronous updates across agent boundaries.
- Using deterministic patterns for workflows that are inherently nondeterministic.
- Using nondeterministic patterns for workflows that are inherently deterministic.
- Ignoring resource constraints when you choose concurrent orchestration.
- Consuming excessive model resources because context windows grow as agents accumulate more information and consult their model to make progress on their task.

Combining orchestration patterns

Applications sometimes require you to combine multiple orchestration patterns to address their requirements. For example, you might use sequential orchestration for the initial data processing stages and then switch to concurrent orchestration for parallelizable analysis tasks. Don't try to make one workflow fit into a single pattern when different stages of your workload have different characteristics and can benefit from each stage using a different pattern.

Relationship to cloud design patterns

AI agent orchestration patterns extend and complement traditional [cloud design patterns](#) by addressing the unique challenges of coordinating intelligent, autonomous components. Cloud design patterns focus on structural and behavioral concerns in distributed systems, but AI agent orchestration patterns specifically address the coordination of components with reasoning capabilities, learning behaviors, and nondeterministic outputs.

Implementations in Microsoft Semantic Kernel

Many of these patterns rely on a code-based implementation to address the orchestration logic. The Agent Framework within Semantic Kernel provides support for many of the following [Agent Orchestration patterns](#):

- [Sequential orchestration](#)
- [Concurrent orchestration](#)
- [Group Chat orchestration](#)
- [Handoff orchestration](#)
- [Magentic orchestration](#)

For hands-on implementation, explore [Semantic Kernel multi-agent orchestration samples](#) ↗ on GitHub that demonstrate these patterns in practice. You can also find many of these patterns in [AutoGen](#) ↗, such as [Magentic-One](#) ↗.

Implementations in Azure AI Foundry Agent Service

You can also use the [Azure AI Foundry Agent Service](#) to chain agents together in relatively simple workflows by using its [connected agents](#) functionality. The workflows that you implement by using this service are primarily nondeterministic, which limits which patterns can be fully implemented in this no-code environment.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal authors:

- [Chad Kittel](#) | Principal Software Engineer - Azure Patterns & Practices
- [Clayton Siemens](#) | Principal Content Developer - Azure Patterns & Practices

Other contributors:

- [Hemavathy Alaganandam](#) | Principal Software Engineer
- [James Lee](#) | Data Scientist 2
- [Ritesh Modi](#) | Principal Software Engineer
- [Mahdi Setayesh](#) | Principal Software Engineer
- [Mark Taylor](#) | Principal Software Engineer
- [Yaniv Vaknin](#) | Senior Technical Specialist

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next step

[Implement agent orchestration with Semantic Kernel](#)

Design and develop a RAG solution

Article • 01/09/2025

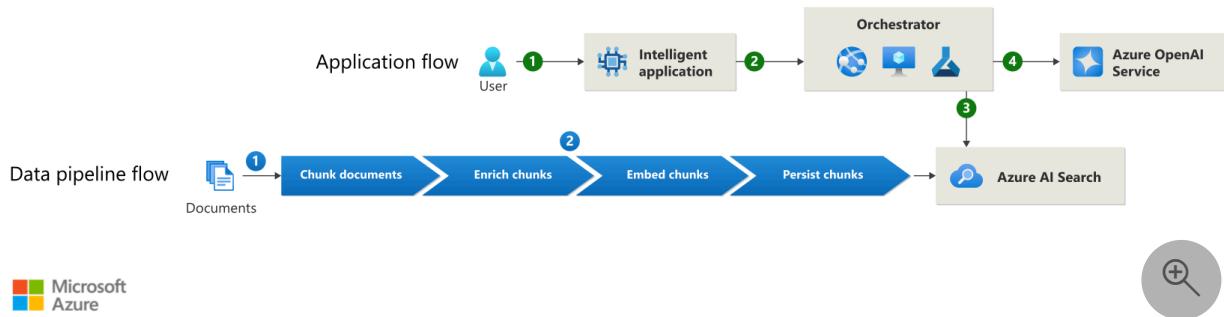
The Retrieval-Augmented Generation (RAG) pattern is an industry-standard approach to building applications that use language models to process specific or proprietary data that the model doesn't already know. The architecture is straightforward, but designing, experimenting with, and evaluating RAG solutions that fit into this architecture involve many complex considerations that benefit from a rigorous, scientific approach.

This article is the introduction of a series. Each article in the series covers a specific phase in RAG solution design.

The other articles in this series cover the following considerations:

- How to determine which test documents and queries to use during evaluation
- How to choose a chunking strategy
- How to determine which chunks you should enrich and how to enrich them
- How to choose the right embedding model
- How to configure the search index
- How to determine which searches, such as vector, full text, hybrid, and manual multiple searches, you should perform
- How to evaluate each step

RAG architecture



RAG application flow

The following workflow describes a high-level flow for a RAG application.

1. The user issues a query in an intelligent application user interface.
2. The intelligent application makes an API call to an orchestrator. You can implement the orchestrator with tools or platforms like Semantic Kernel, Azure Machine

Learning prompt flow, or LangChain.

3. The orchestrator determines which search to perform on Azure AI Search and issues the query.
4. The orchestrator packages the top N results from the query. It packages the top results and the query as context within a prompt and sends the prompt to the language model. The orchestrator returns the response to the intelligent application for the user to read.

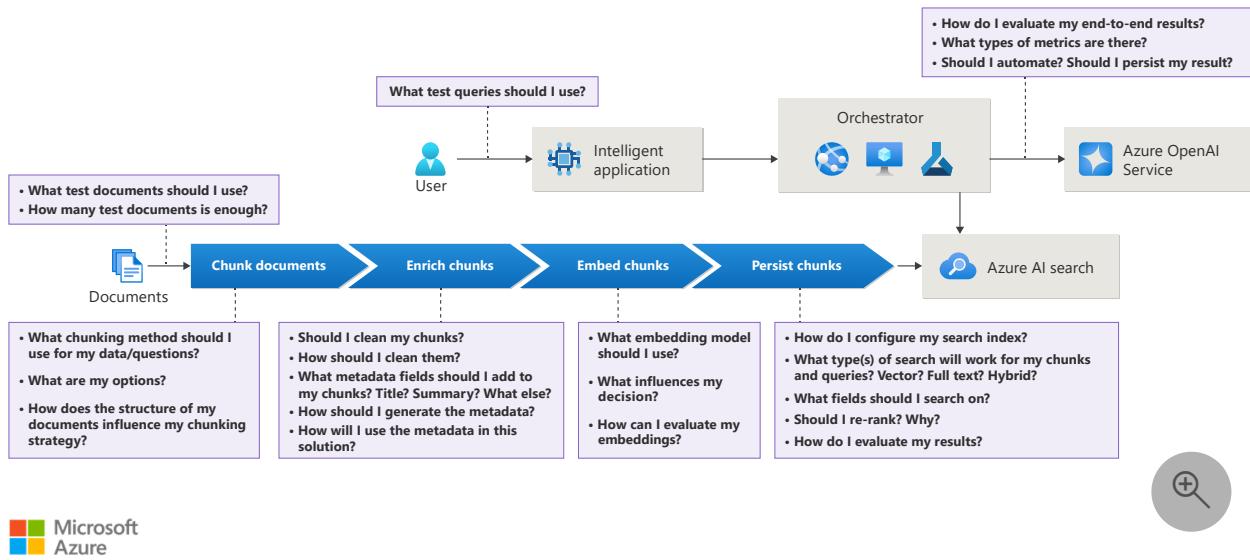
RAG data pipeline flow

The following workflow describes a high-level flow for a data pipeline that supplies grounding data for a RAG application.

1. Documents are either pushed or pulled into a data pipeline.
2. The data pipeline processes each document individually by completing the following steps:
 - a. Chunk document: Breaks down the document into semantically relevant parts that ideally have a single idea or concept.
 - b. Enrich chunks: Adds metadata fields that the pipeline creates based on the content in the chunks. The data pipeline categorizes the metadata into discrete fields, such as title, summary, and keywords.
 - c. Embed chunks: Uses an embedding model to vectorize the chunk and any other metadata fields that are used for vector searches.
 - d. Persist chunks: Stores the chunks in the search index.

RAG design and evaluation considerations

You must make various implementation decisions as you design your RAG solution. The following diagram illustrates some of the questions you should ask when you make those decisions.



The following list provides a brief description of what you should do during each phase of RAG solution development.

- During the **preparation phase**, you should:
 - **Determine the solution domain.** Clearly define the business requirements for the RAG solution.
 - **Gather representative test documents.** Gather test documents for your RAG solution that are representative of your document collection.
 - **Gather test queries.** Gather information and test queries and generate synthetic queries and queries that your documents don't cover.
- During the **chunking phase**, you should:
 - **Understand chunking economics.** Understand which factors to consider as you evaluate the overall cost of your chunking solution for your text collection.
 - **Perform document analysis.** Ask the following questions to help you make decisions when you analyze a document type:
 - What content in the document do you want to ignore or exclude?
 - What content do you want to capture in chunks?
 - How do you want to chunk that content?
 - **Understand chunking approaches.** Understand the different approaches to chunking, including sentence-based, fixed-size, and custom approaches or by using language model augmentation, document layout analysis, and machine learning models.
 - **Understand how document structure affects chunking.** Choose a chunking approach based on the degree of structure that the document has.
- During the **chunk enrichment phase**, you should:
 - **Clean chunks.** Implement cleaning approaches to eliminate differences that don't affect the meaning of the text. This method supports closeness matches.

- **Augment chunks.** Consider augmenting your chunk data with common metadata fields and understand their potential uses in search. Learn about commonly used tools or techniques for generating metadata content.
- During the [embedding phase](#), you should:
 - **Understand the importance of the embedding model.** An embedding model can significantly affect the relevancy of your vector search results.
 - **Choose the right embedding model for your use case.**
 - **Evaluate embedding models.** Evaluate embedding models by visualizing embeddings and calculating embedding distances.
- During the [information retrieval phase](#), you should:
 - **Create a search index.** Apply the appropriate vector search configurations to your vector fields.
 - **Understand search options.** Consider the different types of searches, including vector, full-text, hybrid, and manual multiple searches. Learn about how to split a query into subqueries and filter queries.
 - **Evaluate searches.** Use retrieval evaluation methods to evaluate your search solution.
- During the [language model end-to-end evaluation phase](#), you should:
 - **Understand language model evaluation metrics.** There are several metrics, including groundedness, completeness, utilization, and relevancy, that you can use to evaluate the language model's response.
 - **Understand similarity and evaluation metrics.** You can use similarity and evaluation metrics to evaluate your RAG solution.
 - **Understand the importance of documentation, reporting, and aggregation.** Document the hyperparameters and the evaluation results. Aggregate the results from multiple queries and visualize the results.
 - **Use the RAG experiment accelerator.** Use the [RAG experiment accelerator GitHub repository](#) to help your team find the best strategies for RAG implementation by running multiple experiments, persisting, and evaluating the results.

Structured approach

Because of the number of steps and variables, it's important that you follow a structured evaluation process for your RAG solution. Evaluate the results of each step and make changes based on your requirements. You should evaluate each step independently for optimization, but remember that the end result is what your customers experience.

Make sure that you understand all of the steps in this process before you determine your own acceptance criteria for each step.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [Raouf Aliouat](#) | Software Engineer II
- [Rob Bagby](#) | Principal Architecture Center Content Lead
- [Prabal Deb](#) | Principal Software Engineer
- [Chad Kittel](#) | Principal Software Engineer
- [Ritesh Modi](#) | Principal Engineer
- [Ryan Pfalz](#) | Senior Technical Program Manager
- [Randy Thurman](#) | Principal AI Cloud Solution Architect

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Retrieval Augmented Generation \(RAG\) in Azure AI Search](#)
- [Retrieval augmented generation and indexes](#)

Related resources

Preparation phase

Feedback

Was this page helpful?

 Yes

 No

RAG preparation phase

The first phase of retrieval-augmented generation (RAG) development and experimentation is the preparation phase. During this phase, you define the business domain for your solution. After you define the domain, you gather documents and multimedia content, perform content analysis, and gather sample questions that are pertinent to the domain. You do these steps in parallel because they're interrelated. For example, content analysis helps you determine which test documents, media files, and test queries you should gather. The questions that you ask must be answerable by content in the documents and multimedia, and the content must answer the relevant questions.

This article is part of a series. Read the [introduction](#).

Determine the solution domain

The first step in this process is to clearly define the business requirements for the solution or use case. These requirements help you determine what kind of questions the solution should answer and what source data or documents help answer those questions. In later phases, the solution domain helps inform your embedding model strategy.

Analyze content

The goal of content analysis is to gather enough information about your content collection to help you understand:

- **The different classifications of content.** For example, you might have product specifications, quarterly reports, car insurance contracts, health insurance contracts, training videos, or instructional audio recordings.
- **The different types of content and formats.** For example, you might have PDFs, Markdown files, HTML files, DOCX files, MP4 videos, MP3 audio files, JPEG images, or PowerPoint presentations.
- **The security constraints.** For example, you might require authentication and authorization to access the content depending on whether it's publicly available.
- **The structure and characteristics of the content.** For example, the length of documents might vary, videos might have different durations and resolutions, audio files might contain speech or background music, or content might have subject breaks, contextually relevant images, or tabular data.

The following sections describe how this information helps you choose your loading and chunking strategies.

Understand content classifications

It's important to understand the different content classifications to help determine the number of test items that you need. This part of the analysis should identify high-level classifications, such as insurance or finance. It should also identify subclassifications, such as health insurance documents, car insurance documents, product demonstration videos, or customer service audio recordings. Determine whether these subclassifications have different structures, formats, or content characteristics.

The goal is to understand all of the different content variants that you have. You don't want to overrepresent or underrepresent a specific content classification in your experimentation.

Content types and formats

When you understand the different file formats in your collection, it helps you determine the number and breakdown of test content. For example, if you have PDF and Open XML document types for quarterly reports, you need test content for each of the formats. If you also have video presentations and audio recordings of the same content, include them in your testing. When you know your content types, you can better understand your technical requirements for loading and processing your content. The technical requirements include specific libraries that can process the file formats, transcription services for audio and video content, and computer vision models for image analysis.

Security constraints

You need to understand your security constraints to determine your loading and processing strategies. For example, it's crucial to identify whether some or all of your content requires authentication, authorization, or network visibility. If the content is within a secure perimeter, ensure that your code can access it, or implement a process to securely replicate the content to a location where your processing code can access it.

Documents sometimes reference or embed images, videos, or audio that are important to the context. That media might also be subject to similar access controls as the primary document itself. If the media requires authentication or network line of sight, you must make sure that your code can access the media or that you have a process in place to access and replicate the content. Also, consider privacy and compliance requirements when processing audio and video content that might contain personal data or sensitive conversations.

If your workload requires that different users only have access to distinct content or content segments, ensure that you understand how to retain those access permissions in your chunking solution.

Content structure and characteristics

You need to understand the structure and characteristics of your content, including layout, format, duration, and the types of information contained within. This understanding helps you make the following determinations:

- Whether the content requires preprocessing to clean up noise, extract media, transcribe audio, extract frames from video, reformat content, or annotate items to ignore
- Whether you want to ignore or exclude parts of the content
- What parts of the content you want to capture and process
- How you want to chunk or segment the content
- How you want to handle images, tables, charts, video clips, audio segments, and other embedded media

The following sections list categorized questions that you can use to help you make some of these determinations.

Determine the items that you can ignore

Some structural elements might not add meaning to the document and can safely be ignored when chunking. In some situations, these elements can add valuable context and improve the relevancy of queries to your index, but not all. Ask the following questions about common document features to see whether they add relevancy or should be ignored.

- Does the document contain a table of contents?
- Are there headers or footers?
- Are there copyrights or disclaimers?
- Are there footnotes or endnotes?
- Are there watermarks?
- Are there annotations or comments?

Determine your document preprocessing requirements

The following questions about the structure of the document can help you decide whether you need to preprocess the document to make it easier to process. They also help you choose a chunking strategy.

- Are there multicolumn data or multicolumn paragraphs? You don't want to parse multicolumn content the same way as single-column content.
- How is the document structured? For example, HTML files sometimes use tables that need to be differentiated from embedded tabular data.
- How many paragraphs are there? How long are the paragraphs? Are the paragraphs similar in length?
- What languages, language variants, or dialects are in the documents?
- Does the document contain Unicode characters?
- How are numbers formatted? Do they include commas or decimals? Are they consistent?
- Which parts of the document are uniform, and which parts aren't?
- Is there a header structure where semantic meaning can be extracted?
- Are there bullets or meaningful indentations?

Determine your image preprocessing requirements

To determine your image preprocessing requirements, understand attributes of your images, like whether they have sufficient resolution to process, and whether the image contains all the required information. The following questions can help you determine the requirements.

- What resolution are the images?
- Is there text embedded in the images?
- Are there abstract images that don't add value? For example, icons might not add any semantic value. Adding a description for icons might be detrimental to your solution because the icon usually isn't relevant to the document's content.
- What's the relationship between the images and the surrounding text? Determine whether the images have standalone content or whether there's context around the image that you should use when you pass it to a language model. Captions are an example of surrounding text that might have valuable context that isn't included in the image.

- Is there rich textual representation, such as accessibility descriptions of the images?

Determine your video and audio preprocessing requirements

To determine your video and audio processing requirements, understand your multimedia content, whether it contains valuable information, and how to extract that information effectively. The following questions can help you determine the requirements.

- What are the quality characteristics of the media files? These characteristics include video resolution and framerate, audio quality and sample rate, and file compression and codec types.
- What type of content do the media files contain? Content types include spoken presentations, lectures, interviews, music, sound effects, and screen recordings.
- Do the video files contain visual information that's important for understanding? Examples of the information include presentation slides, demonstrations, charts, graphs, visualizations, text overlays, and captions.
- Are there transcripts or closed captions available for the audio and video content?
- What languages are spoken in the audio and video content?
- Do the media files have metadata or chapters that can help with segmentation?
- Is there background noise or music that might interfere with transcription?
- What's the relationship between multimedia content and accompanying text? Determine whether the media has standalone content or whether there's context in surrounding documents that you should use when processing the media.

Determine your table, chart, and other media-processing requirements

To determine how to process the information encapsulated in tables, charts, and other media, understand its characteristics. The following questions can help you understand your table, chart, and other media-processing requirements.

- Does the document have charts that include numbers?
- Does the document contain tables?
 - Are the tables complex, such as nested tables, or noncomplex?
 - Are there captions for the tables?

- How long are the tables? Long tables might require repeating headers in chunks.
- Are there other types of embedded media, like videos or audio?
- Are there any mathematical equations or scientific notations in the document?

Gather representative test content

In this step, gather content that best represents the content that you use in your solution. The content must address the defined use case and answer the questions that you gathered in the question-gathering parallel phase. The content includes documents, images, videos, audio files, and any other media types that are part of your content collection.

Considerations

Consider the following areas when you evaluate potential representative test content:

- **Pertinence:** The content must meet the business requirements of the conversational application. For example, if you build a chat bot that helps customers perform banking operations, the content must meet that requirement. The content should include documents about opening or closing bank accounts, instructional videos about banking procedures, or audio recordings of customer service interactions. The content must be able to address the test questions that you gather in the parallel step. If the content doesn't have information that's relevant to the questions, your solution can't produce a valid response.
- **Representation:** The content should represent the different types of content that your solution uses. For example, a car insurance document contains different information than a health or life insurance document, and a video demonstration might convey information differently than a written procedure. Suppose that the use case requires the solution to support all three of these insurance types across multiple media formats, but you only have car insurance documents. Your solution might perform poorly for health and life insurance operations or might not use the rich information available in multimedia content. You should have at least two pieces of content for each variation and format.
- **Physical content quality:** The content needs to be in usable shape. Scanned images might not let you extract usable information, poor-quality audio recordings might not transcribe accurately, and low-resolution videos might not provide clear visual information.

- **Content quality:** The content must be high quality. Documents shouldn't contain misspellings or grammatical errors, audio should be clear and audible, and videos should have sufficient resolution and lighting. Language models and other AI services don't perform well if you provide them with poor-quality content.

To successfully gather test content, you should be *qualitatively confident* that the test content fully and accurately represents your specific domain across all media types.

Test content guidance

- Choose real content over synthetic content. Real content must go through a cleaning process to remove personal data, which is especially important for audio and video content that might contain voices, faces, or other identifying information.
- Consider selectively augmenting your content with synthetic data. This process helps you ensure that your content covers all kinds of scenarios, including predicted future scenarios. If you must use synthetic data, do your best to make it resemble real content as much as possible, including realistic audio quality, video characteristics, and visual elements.
- Make sure that the content can address the questions that you gather. Ensure that video content has clear visuals, audio content has clear speech, and multimedia content provides information that complements or supplements text-based content.
- Have at least two pieces of content for each content variant and format. For example, if you have instructional videos, include videos with different presenters, environments, or spoken language.
- Use language models, transcription services, computer vision models, and other tools to help you evaluate the quality of the content across all media types.

Gather test queries

In this step, you gather test queries that you use to evaluate your chunks, your search solution, and your prompt engineering. Do this step while you gather the representative content. Gather the queries and determine how the representative content addresses those queries at the same time. By having both the sample queries and the parts of the sample content that address those queries, you can evaluate every stage of the RAG solution while you experiment with different strategies and approaches.

Gather test query output

The output of this phase includes content from the [gather representative test queries](#) step and the [gather representative test content](#) step. The output is a collection that contains the following data:

- **Query:** The question, which represents a legitimate user's potential prompt.
- **Context:** A collection of the actual text in the documents that address the query. For each bit of context, you should include the page and the actual text.
- **Answer:** A valid response to the query. The response can be content that's directly from the documents, or it might be rephrased from one or more pieces of context.

Create synthetic queries

It's often challenging for the subject matter experts (SMEs) for a particular domain to put together a comprehensive list of questions for the use case. One solution to this challenge is to generate synthetic questions from the representative test documents that you gather. The following steps describe a real-world approach to generating synthetic questions from representative documents:

1. **Chunk the documents.** Break down the documents into chunks. Don't use the chunking strategy for your overall solution. Use this one-off step that you use to generate synthetic queries. You can do the chunking manually if the number of documents is reasonable.
2. **Generate queries for each chunk.** For each chunk, generate queries either manually or by using a language model. When you use a language model, you usually start by generating two queries for each chunk. You can also use the language model to create the answer. The following example shows a prompt that generates questions and answers for a chunk.

text

Please read the following CONTEXT and generate two question and answer JSON objects in an array based on the CONTEXT provided. The questions should require deep reading comprehension, logical inference, deduction, and connecting ideas across the text. Avoid simplistic retrieval or pattern-matching questions. Instead, focus on questions that test the ability to reason about the text in complex ways, draw subtle conclusions, and combine multiple pieces of information to arrive at an answer. Ensure that the questions are relevant, specific, and cover the key points of the CONTEXT. Provide concise answers to each question, and directly quote the text from the provided context. Provide the array output in strict JSON format as shown in the output format. Ensure that the generated JSON is completely structurally correct, including proper nesting, comma placement, and quotation marks. There shouldn't be a comma after the last element in the array.

Output format:

```
[  
  {  
    "question": "Question 1",  
    "answer": "Answer 1"  
  },  
  {  
    "question": "Question 2",  
    "answer": "Answer 2"  
  }  
]
```

CONTEXT:

3. **Verify the output.** Verify that the questions are pertinent to the use case and that the answers address the question. A SME should perform this verification.

Unaddressed queries

It's important to gather queries that the documents don't address and the queries that they do address. When you test your solution, and especially when you test the language model, you need to determine how the solution should respond to queries that it doesn't have sufficient context to answer. To respond to queries that the solution can't address, the solution can:

- State that it doesn't know the answer.
- State that it doesn't know the answer and provide a link where the user might find more information.

Gather test queries for multimedia content

Like with text, you should gather a diverse set of questions that involve using multimedia content to generate highly relevant answers. If you have images with graphs, tables, or screenshots, make sure that you have questions that cover all of the use cases. If you have videos with demonstrations, presentations, or visual information, include questions that require understanding the visual content. For audio content with spoken information, interviews, or presentations, ensure you have questions that test the system's ability to extract and reason about the spoken content.

If you determine in the content analysis step that the text before or after the multimedia is required to answer some questions, make sure that you have those questions in your test queries. Also consider questions that require combining information from multiple media types, such as questions that require understanding both written documents and related instructional videos.

Gather test queries guidance

- Determine whether there's a system that contains real customer questions that you can use. For example, if you build a chat bot to answer customer questions, you might be able to use customer questions from your help desk, FAQs, or ticketing system.
- The customer or SME for the use case should act as a quality gate to determine whether the gathered documents, the associated test queries, and the answers to the queries from the documents are comprehensive, representative, and correct.
- Review the body of questions and answers periodically to ensure that they continue to accurately reflect the source documents.

Next step

[Chunking phase](#)

Related resource

- [Get started with the chat by using your own data sample for Python](#)

Last updated on 11/21/2025

RAG chunking phase

After you gather your test documents and queries and perform a document analysis during the [preparation phase](#), you move to the next phase, which is chunking. Chunking is where you break documents into appropriately sized chunks that each contain semantically relevant content. It's crucial to a successful retrieval-augmented generation (RAG) implementation. If you try to pass entire documents or oversized chunks, it's expensive, might overwhelm the token limits of the model, and doesn't produce the best results. Also, if you pass information to a language model that's irrelevant to the query, it can result in inaccurate or unrelated responses. You must use effective chunking and searching strategies to optimize the process, pass relevant information, and remove irrelevant information. This approach minimizes false positives and false negatives, and maximizes true positives and true negatives.

Chunks that are too small and don't contain sufficient context to address the query can result in poor outcomes. Relevant context that exists across multiple chunks might not be captured. The key is to implement effective chunking approaches for your specific document types and their specific structures and content. There are various chunking approaches to consider, each with their own cost implications and effectiveness, depending on the type and structure of the document that you apply them to.

This article describes various chunking approaches and examines how the structure of your documents can influence the chunking approach that you choose.

This article is part of a series. Read the [introduction](#) before you continue.

Understand chunking economics

When you determine your overall chunking strategy, consider your budget, quality, and throughput requirements for document collection. There are engineering costs for the design and implementation of each unique chunking implementation and per-document processing costs that differ depending on the approach. If your documents contain embedded or linked media, you must consider the economics of processing those elements. For chunking, this processing generally uses language models to generate descriptions of the media. Those descriptions are then chunked. An alternative approach for some media is to pass them as-is to a multimodal model at inferencing time. But this approach doesn't affect the chunking economics.

The following sections examine the economics of chunking images and the overall solution.

Understand image chunking economics

A language model that generates a description of an image that you chunk incurs extra cost. For example, cloud-based services such as Azure OpenAI Service either charge on a per-transaction basis or on a prepaid provisioning basis. Larger images incur a larger cost. Through your document analysis, you determine which images are valuable to chunk and which images to ignore. From there, you need to understand the number and sizes of the images in your solution. You then weigh the value of chunking the image descriptions against the cost to generate those descriptions.

Use a service such as [Azure AI Vision](#) to determine which images you want to process. You can classify images, tag images, or do logo detection. You can use the results and confidence indicators to determine whether the image adds meaningful, contextual value and should be processed. Calls to Vision might be less expensive than calls to language models, so this approach could result in cost savings. Experiment to determine what confidence levels and classifications or tags provide the best results for your data. Also consider the following alternatives:

- Build your own classifier model. If you take this approach, be sure you consider the costs to build, host, and maintain your own model.
- Use a language model to classify and tag images. This approach gives you more flexibility in your design but can be less predictable than using Vision. When possible, experiment with both approaches and compare the results.

Another cost optimization strategy is to cache by using the [Cache-Aside pattern](#). You can generate a key that you base on the hash of the image. As a first step, check to see if you have a cached result from a prior run or previously processed document. If you do, you can use that result. This approach eliminates the costs of calling a classifier or a language model. If there's no cache, when you call to the classifier or language model, you cache the result. Future calls for this image use the cache.

The following workflow integrates these cost optimization processes:

1. Check to see if the image processing is cached. If so, use the cached results.
2. Run your classifier to determine whether you should process the image. Cache the classification result. If your classification logic determines that the image adds value, proceed to the next step.
3. Generate the description for your image. Cache the result.

Consider the economics of your overall solution

Consider the following factors when you assess the cost of your overall solution:

- **Number of unique chunking implementations:** Each unique implementation has engineering and maintenance costs. Consider the number of unique document types in your collection and the cost versus quality trade-offs of unique implementations for each.
- **Per-document cost of each implementation:** Some chunking approaches might result in better quality chunks but have a higher financial and temporal cost to generate those chunks. For example, using a prebuilt model in Azure AI Document Intelligence likely has a higher per-document cost than a pure text parsing implementation, but might result in better chunks.
- **Number of initial documents:** The number of initial documents that you need to process to launch your solution.
- **Number of incremental documents:** The number and rate of new documents that you must process for ongoing maintenance of the system.

Understand loading and chunking

During chunking, you must first load the document into memory in some format. The chunking code then operates against the in-memory representation of the document. You can combine the loading code with chunking, or separate loading into its own phase. Choose an approach based on architectural constraints and your preferences. The following sections briefly explore both options and provide general recommendations.

Separate loading and chunking

There are several reasons why you would choose to separate the loading and chunking phases. You might want to encapsulate logic in the loading code. You might want to persist the result of the loading code before chunking, especially when you experiment with various chunking permutations to save on processing time or cost. Lastly, you might want to run the loading and chunking code in separate processes for architectural reasons, such as process bulkheading or security segmentation that involves removing personal data.

Encapsulate logic in the loading code

You can choose to encapsulate preprocessing logic in the loading phase. This approach simplifies the chunking code because it doesn't require any preprocessing. Preprocessing can be as simple as removing or annotating parts of the document that you want to ignore in document analysis. For example, you might want to remove watermarks, headers, and footers. Or preprocessing can involve more complex tasks such as reformatting the document. For example, you can include the following preprocessing tasks in the loading phase:

- Remove or annotate items that you want to ignore.
- Replace image references with image descriptions. During this phase, you use a large language model to generate a description for an image and update the document with that description. If during document analysis, you find surrounding text that provides valuable context, you can pass the text and the image to the large language model.
- Download or copy images to file storage like Azure Data Lake Storage to be processed separately from the document text. If during document analysis, you find surrounding text that provides valuable context to the image, you can store this text along with the image in file storage.
- Reformat tables so that they're more easily processed.
- Define document structure based on headings, subheadings, and other structural elements. Use a tool like [Document Intelligence](#) when practical to reduce development overhead. Or you can use a library like Python if you have sensitive data that you can't send to an external system.

Persist the result of the loading code

There are multiple reasons why you might choose to persist the result of the loading code. If you want the ability to inspect the documents after they're loaded and preprocessed, but before the chunking logic runs. Or you want to run different chunking logic against the same preprocessed code while it's in development or in production. Persisting the loaded code speeds up the process.

Run loading and chunking code in separate processes

When you separate the processes, it helps you run multiple chunking implementations against the same preprocessed code. You can also run loading and chunking code in different compute environments and on different hardware. This design helps you independently scale the compute that you use for loading and chunking.

Combine loading and chunking

In most cases, combining your loading and chunking code is a simpler implementation. Many preprocessing operations that you might consider doing in a separate loading phase can run during the chunking phase. For example, instead of replacing image URLs with a description in the loading phase, the chunking logic can make calls to the large language model to get a text description and chunk the description.

When you have document formats like HTML that contain tags with references to images, ensure that the reader or parser that the chunking code uses doesn't remove the tags. The chunking code must be able to identify image references.

Consider the chunking recommendations

Consider the following recommendations on whether to combine or separate your chunking logic.

- Start by combining your loading and chunking logic. Separate them when your solution requires it.
- Avoid converting documents to an intermediate format if you choose to separate the processes. This type of operation can result in data loss.

Review the chunking approaches

This section provides an overview of common chunking approaches. You can use multiple approaches in implementation, such as combining the use of a language model to get a text representation of an image with many of the listed approaches.

A summarized decision-making matrix accompanies each approach. The matrix highlights the tools, associated costs, and more. The engineering effort and processing costs described here are subjective and included for relative comparison.

Important

Your chunking approach is a semipermanent choice in your overall solution design. Do a thorough comparison of the approaches to find the best fit for your use case and content type prior to choosing one to use in production. When you change chunking strategies, it can significantly affect downstream processes and require changes throughout the workflow.

Fixed-size parsing, with overlap

This approach breaks down a document into chunks based on a fixed number of characters or tokens and allows overlap of characters between chunks. This approach has many of the same advantages and disadvantages as sentence-based parsing. One advantage of this approach over sentence-based parsing is the ability to obtain chunks with semantic meanings that span multiple sentences.

You must choose the fixed size of the chunks and the amount of overlap. Because the results vary for different document types, it's best to use a tool like the Hugging Face chunk visualizer to do exploratory analysis. You can use tools like this to visualize how your documents are chunked based on your decisions. You should use bidirectional encoder representations from transformers (BERT) tokens instead of character counts when you use fixed-sized parsing. BERT tokens are based on meaningful units of language, so they preserve more semantic information than character counts.

Tools: [LangChain recursive text splitter](#), [Hugging Face chunk visualizer](#)

Engineering effort: Low

Processing cost: Low

Use cases: Unstructured documents written in prose or nonprose with complete or incomplete sentences. Your collection of documents contains a prohibitive number of different document types that require individual chunking strategies.

Examples: User-generated content like open-ended feedback from surveys, forum posts, reviews, email messages, personal notes, research notes, and lists

Semantic chunking

This approach uses embeddings to group conceptually similar content across a document to create chunks. Semantic chunking can produce easily understandable chunks that closely align to the content's subjects. The logic for this approach can search a document or set of documents to find recurring information and create chunks that group the mentions or sections together. This approach can be more costly because it requires you to develop complex custom logic.

Tools: Custom implementation. Natural language processing (NLP) tools like spaCy can help with sentence-based parsing.

Engineering effort: High

Processing cost: High

Use cases: Documents that have topical overlap throughout their sections

Examples: Financial or healthcare-focused documentation

Custom code

This approach parses documents by using custom code to create chunks. The custom code approach works best for text-based documents where the structure is known or can be inferred. The custom code approach requires a high degree of control over chunk creation. You can use text parsing techniques like regular expressions to create chunks based on patterns within the document's structure. The goal is to create chunks that have similar size in length and chunks that have distinct content. Many programming languages provide support for

regular expressions, and some have libraries or packages that provide more elegant string manipulation features.

Tools: Python ([re](#), [regex](#), [BeautifulSoup](#), [lxml](#), [html5lib](#), [marko](#)), R ([stringr](#), [xml2](#)), Julia ([Gumbo](#))

Engineering effort: Medium

Processing cost: Low

Use cases: Semi-structured documents where structure can be inferred

Examples: Patent filings, research papers, insurance policies, scripts, and screenplays

Language model augmentation

You can use language models to create chunks. For example, use a large language model, such as GPT-4, to generate textual representations of images or summaries of tables that become chunks. Language model augmentation is often used with other chunking approaches such as custom code.

If your document analysis determines that the text before or after the image helps [answer some requirement questions](#), pass this extra context to the language model. It's important to experiment to determine whether this extra context improves the performance of your solution.

If your chunking logic splits the image description into multiple chunks, include the image URL in each chunk to ensure that metadata is returned for all queries that the image serves. This step is crucial for scenarios where the user needs to access the source image through that URL or use raw images during inferencing time.

Tools: Azure OpenAI, OpenAI

Engineering effort: Medium

Processing cost: High

Use cases: Images, tables

Examples: Generate text representations of tables and images, summarize transcripts from meetings, speeches, interviews, or podcasts

Document layout analysis

Document layout analysis libraries and services combine optical character recognition (OCR) capabilities with deep learning models to extract both the structure and text of documents. Structural elements can include headers, footers, titles, section headings, tables, and figures. The goal is to provide better semantic meaning to content that the documents contain.

Document layout analysis libraries and services expose a model that represents the structural and textual content of the document. You still have to write code that interacts with the model.

! Note

Document Intelligence is a cloud-based service that requires you to upload your document. You must ensure that your security and compliance regulations enable you to upload documents to such services.

Tools: [Document Intelligence document analysis models](#), [Donut](#), [Layout Parser](#)

Engineering effort: Medium

Processing cost: Medium

Use cases: Semi-structured documents

Examples: News articles, web pages, and resumes

Graph-based chunking

Graph-based chunking is an iterative approach that involves using a language model to find entities, like keywords, in documents and to build a graph based on their relationships. You can start with a simpler chunking strategy, like paragraph-based, to make the graph-building process more efficient. After you have your initial chunks, you can analyze each one to find entities and relationships and build your global graph structure. You can append the graph as you iterate through the chunks.

Tools: [Microsoft GraphRAG](#), Neo4J

Engineering effort: High

Processing cost: High

Use cases: Diverse statistical data

Examples: Sports analytics, historical data, and any domain that requires unplanned quantitative queries across documents

Prebuilt model

Services such as Document Intelligence provide prebuilt models that you can use for various document types. Some models are trained for specific document types, such as the U.S. W-2 tax form, while others target a broader genre of document types such as invoices.

Tools: [Document Intelligence prebuilt models](#), [Power Automate intelligent document processing](#), [LayoutLMv3](#)

Engineering effort: Low

Processing cost: Medium/High

Use cases: Structured documents where a prebuilt model exists

Examples: Invoices, receipts, health insurance cards, and W-2 forms

Custom model

For highly structured documents where no prebuilt model exists, you might have to build a custom model. This approach can be effective for images or documents that are highly structured, which makes using text parsing techniques difficult.

Tools: [Document Intelligence custom models](#), [Tesseract](#) ↗

Engineering effort: High

Processing cost: Medium/High

Use cases: Structured documents where a prebuilt model doesn't exist

Examples: Automotive repair and maintenance schedules, academic transcripts, records, technical manuals, operational procedures, and maintenance guidelines

Sentence-based parsing

Sentence-based parsing is a straightforward approach that breaks text documents into chunks, which are composed of complete sentences. Use this approach as a fallback solution if none of the other approaches described here fit your use case. The advantages of this approach include its low implementation and processing costs and its applicability to any text-based document that contains prose or full sentences. One drawback of this approach is that each chunk might not capture the full context of an idea or meaning. Multiple sentences must often be taken together to capture the semantic meaning.

Tools: [spaCy sentence tokenizer](#) ↗, [LangChain recursive text splitter](#) ↗, [NLTK sentence tokenizer](#) ↗

Engineering effort: Low

Processing cost: Low

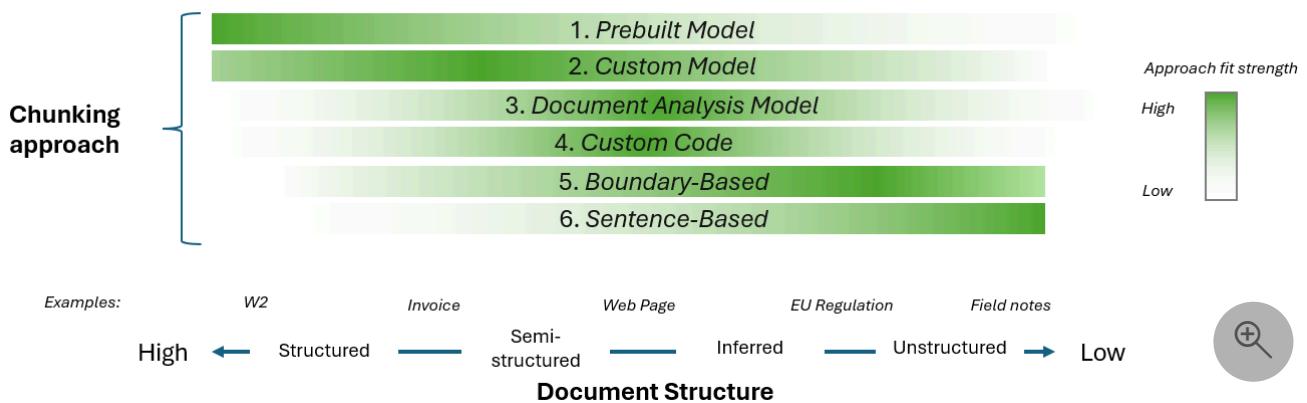
Use cases: Unstructured documents written in prose or full sentences. Your collection of documents contains a prohibitive number of different document types, which require individual chunking strategies

Examples: User-generated content like open-ended feedback from surveys, forum posts, reviews, email messages, novels, or essays

Document structure

Documents vary in their type of structure. Some documents, like government forms, have a complex and well-known structure, such as a U.S. W-2 tax form. At the other end of the spectrum are unstructured documents like free-form notes. The degree of structure in a

document type is a good starting point to determine an effective chunking approach. While there are no specific rules, this section provides you with some guidelines to follow.



Structured documents

Structured documents, sometimes referred to as fixed-format documents, contain defined layouts. The data in these documents is located at fixed locations. For example, the date, or customer family name, is in the same location of every document that has the same fixed format.

Fixed-format documents might be scanned images of original documents that are hand-filled or have complex layout structures. This format makes them difficult to process by using a basic text parsing approach. A typical approach to processing complex document structures is to use machine learning models to extract data and apply semantic meaning to that data, when possible.

Examples: W-2 form and insurance card

Typical approaches: Prebuilt models and custom models

Semi-structured documents

Semi-structured documents don't have a fixed format or schema, like the W-2 form, but they provide consistency regarding format or schema. For example, invoices vary in layout, but they generally have a consistent schema. You can expect an invoice to have an *invoice number* and some form of *bill to* and *ship to* name and address, among other data. A web page might not have schema consistencies, but they have similar structural or layout elements, such as *body*, *title*, *H1*, and *p* that can add semantic meaning to the surrounding text.

Like structured documents, semi-structured documents that have complex layout structures are difficult to process by using text parsing. For these document types, machine learning models are a good approach. There are prebuilt models for certain domains that have consistent

schemas like invoices, contracts, or health insurance documents. Consider building custom models for complex structures where no prebuilt model exists.

Examples: Invoices, receipts, web pages, and Markdown files

Typical approaches: Document analysis models

Inferred structure

Some documents have a structure but aren't written in markup. For these documents, the structure must be inferred. A good example is the following EU regulation document.

CHAPTER I

General provisions

Article 1

Subject matter

1. In order to achieve a high common level of digital operational resilience, this Regulation lays down uniform requirements concerning the security of network and information systems supporting the business processes of financial entities as follows:

(a) requirements applicable to financial entities in relation to:

(i) information and communication technology (ICT) risk management;

(ii) reporting of major ICT-related incidents and notifying, on a voluntary basis, significant cyber threats to the competent authorities;

(iii) reporting of major operational or security payment-related incidents to the competent authorities by financial entities referred to in Article 2(1), points (a) to (d);

(iv) digital operational resilience testing;

(v) information and intelligence sharing in relation to cyber threats and vulnerabilities;

(vi) measures for the sound management of ICT third-party risk;

(b) requirements in relation to the contractual arrangements concluded between ICT third-party service providers and financial entities;

(c) rules for the establishment and conduct of the Oversight Framework for critical ICT third-party service providers when providing services to financial entities;

(d) rules on cooperation among competent authorities, and rules on supervision and enforcement by competent authorities in relation to all matters covered by this Regulation.

2. In relation to financial entities identified as essential or important entities pursuant to national rules transposing Article 3 of Directive (EU) 2022/2555, this Regulation shall be considered a sector-specific Union legal act for the purposes of Article 4 of that Directive.

3. This Regulation is without prejudice to the responsibility of Member States' regarding essential State functions concerning public security, defence and national security in accordance with Union law.



Because you can clearly understand the structure of the document, and there are no known models for it, you must write custom code. This document format might not warrant the effort to create a custom model, depending on the number of different documents of this type that you work with. For example, if your collection contains all EU regulations or U.S. state laws, a custom model might be a good approach. If you work with a single document, like the EU regulation in the example, custom code might be more cost effective.

Examples: Law documents, scripts, and manufacturing specifications

Typical approaches: Custom code and custom models

Unstructured documents

A good approach for documents that have little to no structure are sentence-based or fixed-size with overlap.

Examples: User-generated content like open-ended feedback from surveys, forum posts, reviews, email messages, personal notes, and research notes

Typical approaches: Sentence-based or boundary-based with overlap

Experimentation

This article describes the most suitable chunking approaches for each document type, but in practice, any of the approaches might be appropriate for any document type. For example, sentence-based parsing might be appropriate for highly structured documents, or a custom model might be appropriate for unstructured documents. Part of optimizing your RAG solution is to experiment with various chunking approaches. Consider the number of resources that you have, the technical skill of your resources, and the volume of documents that you need to process. To achieve an optimal chunking strategy, test each approach and observe the advantages and trade-offs to ensure that you choose the best approach for your use case.

Next step

Chunk enrichment phase

Related resources

- [Chunking large documents for vector search solutions in Azure AI Search](#)
- [Integrated data chunking and embedding in Azure AI Search](#)

Last updated on 11/21/2025

RAG chunk enrichment phase

After you break your documents into a collection of chunks, the next step is to enrich each chunk by cleaning it and augmenting it with metadata. Cleaning the chunks enables you to achieve better matches for semantic queries in a vector search. Adding metadata enables you to support searches of the chunks that go beyond semantic searches. Both cleaning and augmenting involve extending the schema for the chunk.

This article discusses various ways to augment your chunks, including some common cleaning operations that you can perform on chunks to improve vector comparisons. It also describes some common metadata fields that you can add to your chunks to augment your search index.

ⓘ Note

This article focuses only on vector-based retrieval-augmented generation (RAG) solutions. Strategies related to graph-based, agentic, tag-augmented generation (TAG), and other RAG solutions aren't in scope.

This article is part of a series. Read the [introduction](#).

The following image shows a code sample of chunks that are enriched with data.

```
[{"Chunk": "Lorum ipsum ..."}, {"Chunk": "Duis aute ..."}, {"CleanedChunk": "lorum ipsum ..."}, {"Title": "", "Summary": "", "keywords": [""], "questions": [""]}, {"Chunk": "Duis aute ..."}, {"CleanedChunk": "duis aute ..."}, {"Title": "", "Summary": "", "keywords": [""], "questions": [""]}]
```

Clean your data

When you clean your data, it helps your workload find the most relevant chunks, typically through vectorizing those chunks and storing them in a vector database. An optimized vector search returns only the rows in the database that have the closest semantic matches to the query. The goal of cleaning the data is to support closeness matches by eliminating potential differences that aren't material to the semantics of the text.

ⓘ Note

To return the original, uncleaned chunk as the query result, add an extra field to store the cleaned and vectorized data.

Consider the following common cleaning procedures:

- **Implement lowercasing strategies.** Lowercasing allows words that are capitalized, such as words at the beginning of a sentence, to match corresponding words within a sentence. Embeddings are typically case-sensitive, so "Cheetah" and "cheetah" would result in a different vector for the same logical word. For example, for the embedded query "what is faster, a cheetah or a puma?" the embedding "cheetahs are faster than pumas" is a closer match than "Cheetahs are faster than pumas." Some lowercasing strategies lowercase all words, including proper nouns, while other strategies lowercase only the first words in sentences.
- **Guard against prompt injection attacks.** If an attacker knows that a content repository is processed for indexing, the attacker can try to add content to the repository that includes instructions for your language models and agents to evaluate. Never use the data from your content as a source of instructions during any processing tasks.

Improve security with techniques like flagging or excluding media that contains embedded instructions. Consider using a language model with constrained decoding to classify and sanitize inputs.

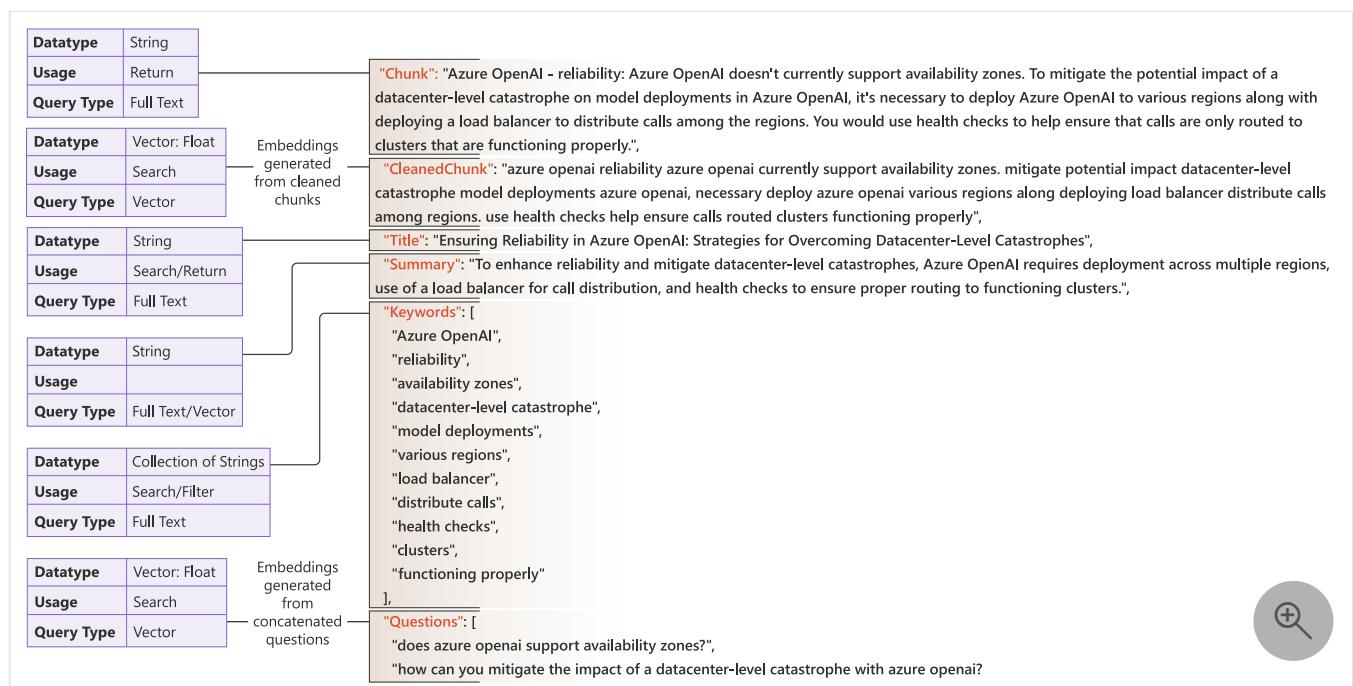
- **Remove stop words.** Stop words are words like "a," "an," and "the." You can remove stop words to reduce the dimensionality of the resulting vector. If you remove stop words in the previous example, "a cheetah is faster than a puma," and "the cheetah is faster than the puma," are vectorial equal to "cheetah faster puma." But it's important to understand that some stop words hold semantic meaning. For example, "not" might be considered a stop word, but it holds significant semantic meaning. You need to test to determine the effect of removing stop words.

- **Fix spelling mistakes.** A misspelled word doesn't match with the correctly spelled word in the embedding model. For example, "cheataah" isn't the same as "cheetah" in the embedding. You should fix spelling mistakes to address this problem.
- **Remove Unicode characters.** Remove Unicode characters to reduce noise in your chunks and reduce dimensionality. Like stop words, some Unicode characters might contain relevant information. It's important to conduct testing to understand the implications of removing Unicode characters.
- **Normalize text.** Normalize text according to standards like expanding abbreviations, converting numbers to words, and expanding contractions, for example, expanding "I'm" to "I am." This approach can help improve the performance of vector searches.
- **Normalize localization.** Prefer localizing at the document level and reprocessing each language separately. Avoid storing unvalidated translations. Ensure that your embedding model supports multilingual input.

Augment your chunks

Semantic searches against the vectorized chunks work well for some types of queries but not as well for others. Depending upon the query types that you need to support, you might need to augment your chunks with extra information. The extra metadata fields are all stored in the same row as your embeddings and can be used in the search solution either as filters or as part of a search.

The following image shows the JSON of fully enriched content and describes how a search platform might use the metadata.



The metadata columns that you need to add depend on your problem domain, including the type of data you have and the types of queries you want to support. You need to analyze the user experience, available data, and result quality you're trying to achieve. From there, you can determine what metadata might help you address your workload's requirements.

The following list describes common metadata fields, the original chunk text, potential uses, and tools or techniques that generate the metadata content.

- **ID.** An ID uniquely identifies a chunk. A unique ID is useful during processing to determine whether a chunk already exists in the store. An ID can be a hash of some key field. **Tools:** A hashing library.
- **Title.** A title is a useful return value for a chunk. It provides a quick summary of the content in the chunk. The summary can also be useful to query with an indexed search because it can contain keywords for matching. **Tools:** A language model.
- **Summary.** The summary is similar to the title in that it's a common return value and can be used in indexed searches. Summaries are often longer than titles. **Tools:** A language model.
- **Rephrasing of the chunk.** Rephrasing of a chunk can be helpful as a vector search field because rephrasing captures variations in language, such as synonyms and paraphrasing. **Tools:** A language model.
- **Keywords.** Keyword searches are useful for data that's noncontextual, for searching for an exact match, and when a specific term or value is important. For example, an auto manufacturer might have reviews or performance data for each of its models for multiple years. "Review for product X for year 2009" is semantically like "Review for product X for 2010" and "Review for product Y for 2009." In this case, it's more effective to match on keywords for the product and year. **Tools:** A language model, RAKE, KeyBERT, multi-rake.
- **Tags.** Tags can be keywords or classifiers, like Multipurpose Internet Mail Extensions (MIME) types. Using tags is helpful for hybrid search (vector + text) and advanced filtering. **Tools:** A language model.
- **Entities.** Entities are specific pieces of information, like people, organizations, and locations. Like keywords, entities are good for exact match searches or when specific entities are important. **Tools:** spaCy, Stanford Named Entity Recognizer (Stanford NER), scikit-learn, and Natural Language Toolkit (NLTK).
- **Cleaned chunk text.** The text of the cleaned chunk. **Tools:** A language model.
- **Questions that the chunk can answer.** Sometimes, the embedded query isn't a close match to the embedded chunk. For example, the query might be small relative to the

chunk size. It might be better to formulate the queries that the chunk can answer and do a vector search between the user's actual query and the preformulated queries. **Tools:** A language model.

- **Source.** The source of the chunk can be valuable as a return for queries. Returning the source allows the querier to cite the original source.
- **Language.** The language of the chunk can be useful as a filter in queries.

Consider multimodal enrichment recommendations

When you work with video, image, or audio media, consider the following recommendations.

- Generate descriptive text for each artifact and include localized translations.
- Generate multiple representations for each artifact.

Calculate the cost of augmenting

Some language models that augment chunks can be expensive. You need to calculate the cost of each enrichment that you're considering and multiply it by the estimated number of chunks over time. Use this information, along with your testing of the enriched fields, to determine the best business decision.

Next step

[Generate embeddings](#)

Related resources

- [AI enrichment in Azure AI Search](#)
- [Skill set concepts in Azure AI Search](#)

Last updated on 11/21/2025

RAG generate embeddings phase

In the previous steps of your retrieval-augmented generation (RAG) solution, you divided your documents into chunks and enriched the chunks. In this step, you generate embeddings for those chunks and any metadata fields on which you plan to perform vector searches.

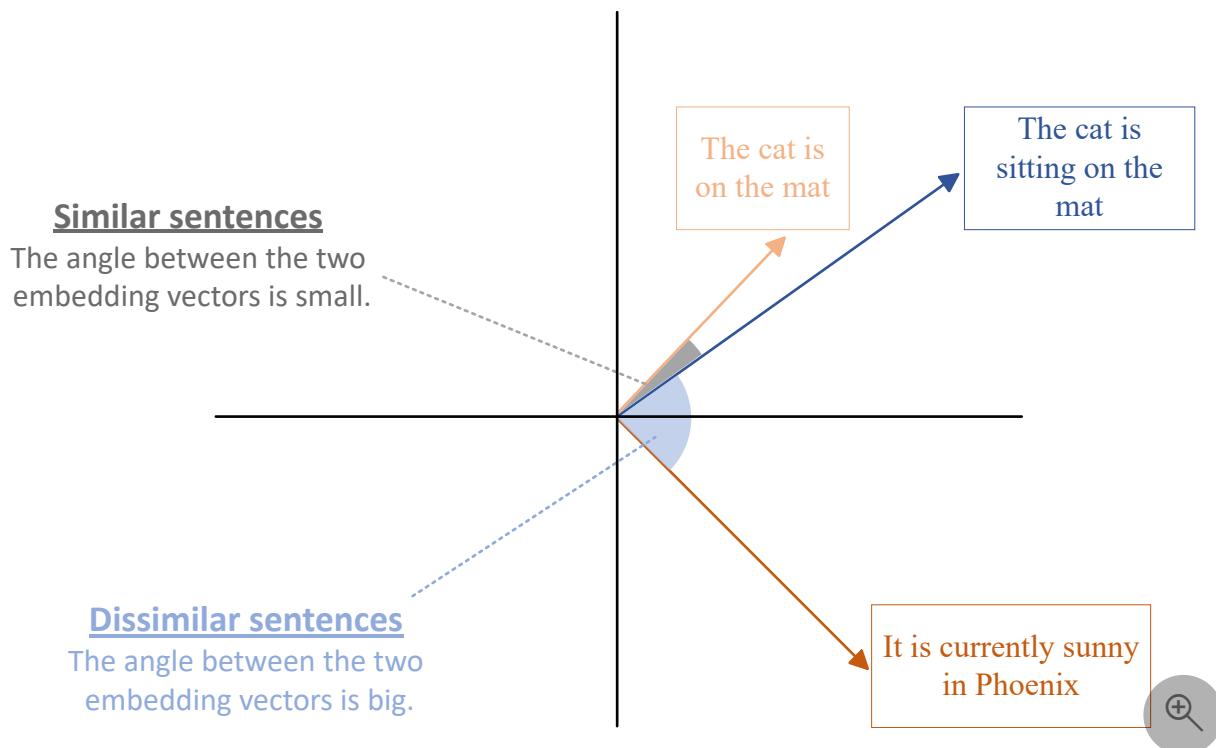
This article is part of a series. Read the [introduction](#).

An embedding is a mathematical representation of an object, such as text. When a neural network is being trained, the process creates many representations of an object. Each representation has connections to other objects in the network. An embedding is important because it captures the semantic meaning of the object.

The representation of one object has connections to representations of other objects, so you can compare objects mathematically. The following example shows how embeddings capture semantic meaning and relationships between objects:

```
embedding (king) - embedding (man) + embedding (woman) = embedding (queen)
```

Embeddings are compared to one another by using the notions of similarity and distance. The following grid shows a comparison of embeddings.



In a RAG solution, you embed the user query by using the same embedding model as your chunks. Then, you search your database for relevant vectors to return the most semantically relevant chunks. The original text of the relevant chunks passes to the language model as grounding data.

! Note

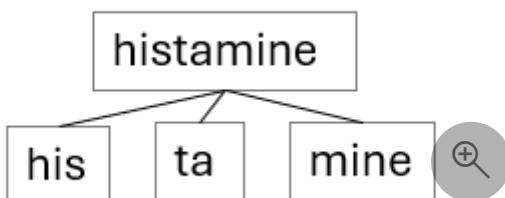
Vectors represent the semantic meaning of text in a way that allows for mathematical comparison. You must clean the chunks so that the mathematical proximity between vectors accurately reflects their semantic relevancy.

Understand the importance of your embedding model

The embedding model that you choose can significantly affect the relevancy of your vector search results. You must carefully consider the vocabulary of the embedding model. Every embedding model is trained with a specific vocabulary. For example, the vocabulary size of the [bidirectional encoder representations from transformers \(BERT\) model ↗](#) is about 30,000 words.

The vocabulary of an embedding model is important because it handles words that aren't in its vocabulary in a unique manner. If a word isn't in the model's vocabulary, it still calculates a vector for it. Many models break down the words into subwords. They treat the subwords as distinct tokens, or they aggregate the vectors for the subwords to create a single embedding.

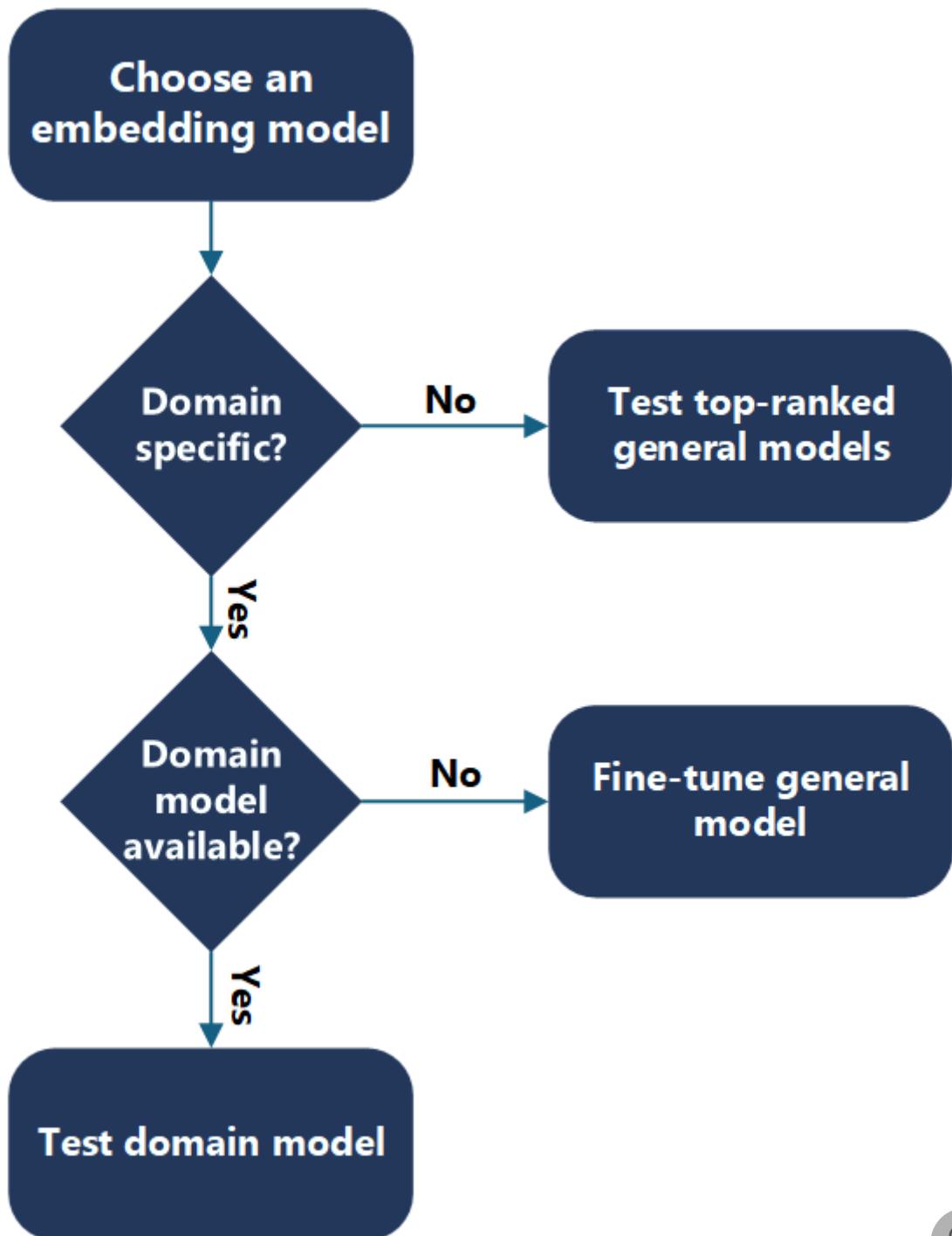
For example, the word *histamine* might not be in an embedding model's vocabulary. The word *histamine* has a semantic meaning as a chemical that your body releases, which causes allergy symptoms. The embedding model doesn't contain *histamine*. So, it might separate the word into subwords that are in its vocabulary, such as *his*, *ta*, and *mine*.



The semantic meanings of these subwords are far from the meaning of *histamine*. The individual or combined vector values of the subwords result in poorer vector match compared to if the word *histamine* were in the model's vocabulary.

Choose an embedding model

Determine the right embedding model for your use case. Consider the overlap between the embedding model's vocabulary and your data's words when you choose an embedding model.



First, determine whether you have domain-specific content. For example, are your documents specific to a use case, your organization, or an industry? A good way to determine domain specificity is to check whether you can find the entities and keywords in your content on the internet. If you can, a general embedding model likely can, too.

General or non-domain-specific content

When you choose a general embedding model, start with the [Hugging Face leaderboard ↗](#) model rankings. Evaluate how the models work with your data, and start with the top-ranking

models.

Domain-specific content

For domain-specific content, determine whether you can use a domain-specific model. For example, your data might be in the biomedical domain, so you might use the [BioGPT model](#). This language model is pretrained on a large collection of biomedical literature. You can use it for biomedical text mining and generation. If domain-specific models are available, evaluate how these models work with your data.

If you don't have a domain-specific model, or the domain-specific model doesn't perform well, you can fine-tune a general embedding model with your domain-specific vocabulary.

 **Important**

For any model that you choose, you need to verify that the license suits your needs and the model provides the necessary language support.

Generate multimodal embeddings

Embeddings aren't limited to text. You can generate embeddings for text and other media types, such as images, audio, and video. The process for generating embeddings is similar across modalities. Load the content, pass it through the embedding model, and store the resulting vector. But the choice of model and preprocessing steps vary by media type.

For example, you can use models like [Contrastive Language–Image Pre-training \(CLIP\)](#) to generate image embeddings. You can then use the embeddings in vector search to retrieve semantically similar images. For video, you must define a schema that extracts specific features, like object presence or narrative summary and use specialized models to generate embeddings for those features.

 **Tip**

Use a schema to define the features that you want to extract from multimodal content. This approach optimizes your embeddings for your retrieval goals.

Use dimensionality reduction

Embedding vectors can be high dimensional, which increases storage and compute costs. Dimensionality reduction techniques help make embeddings more manageable, cost effective, and interpretable.

You can use algorithms such as [t-distributed stochastic neighbor embedding \(t-SNE\)](#) or [principal component analysis \(PCA\)](#) to reduce the number of dimensions in your vectors.

These tools are available in libraries like PyTorch and scikit-learn.

Dimensionality reduction can improve semantic clarity and visualization. It also helps eliminate unused or noisy features in dense embeddings.

!Note

Dimensionality reduction is a post-processing step. You apply it after generating embeddings to optimize storage and retrieval performance.

Compare embeddings

When you evaluate embeddings, you can use mathematical formulas to compare vectors. These formulas help you determine how similar or dissimilar two embeddings are.

Common comparison methods include:

- **Cosine similarity:** Measures the angle between two vectors. Useful for high-dimensional data.
- **Euclidean distance:** Measures the straight-line distance between two vectors.
- **Manhattan distance:** Measures the absolute difference between vector components.
- **Dot product:** Measures the projection of one vector onto another.

💡 Tip

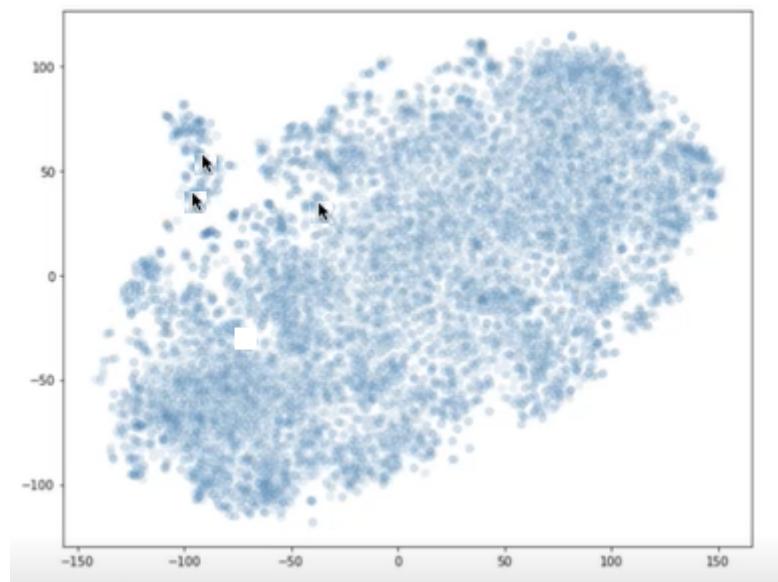
Choose your comparison method based on your use case. For example, use cosine similarity when you want to measure semantic closeness, and use Euclidean distance when you want to measure literal proximity.

Evaluate embedding models

To evaluate an embedding model, visualize the embeddings and evaluate the distance between the question and chunk vectors.

Visualize embeddings

You can use libraries, such as t-SNE, to plot the vectors for your chunks and your question on an X-Y graph. You can then determine how far the chunks are from one another and from the question. The following graph shows chunk vectors plotted. The two arrows near one another represent two chunk vectors. The other arrow represents a question vector. You can use this visualization to understand how far the question is from the chunks.



Calculate embedding distances

You can use a programmatic method to evaluate how well your embedding model works with your questions and chunks. Calculate the distance between the question vectors and the chunk vectors. You can use the Euclidean distance or the Manhattan distance.

Evaluate embedding models by using retrieval performance

To choose the best embedding model, evaluate how well it performs in retrieval scenarios. Embed your content, perform vector search, and assess whether the correct items are retrieved.

You can experiment with different models, comparison formulas, and dimensionality settings. Use evaluation metrics to determine which model provides the best results for your use case.

i Important

Retrieval performance is the most practical way to evaluate embedding quality. Use real-world queries and content to test your models.

Fine-tune embedding models

If a general or domain-specific model doesn't meet your needs, you can fine-tune it with your own data. Fine-tuning adjusts the model's weights to better represent your vocabulary and semantics.

Fine-tuning can improve retrieval accuracy, especially for specialized domains like code search or legal documents. But it requires careful evaluation and can sometimes degrade performance if the training data is poor.

Modern techniques like one-bit reinforcement learning (RL) make your fine-tuning more cost effective.

💡 Tip

Before you fine-tune your model, evaluate whether prompt engineering or constrained decoding can solve your problem. Use evaluation metrics and retrieval performance to guide your fine-tuning.

Use the Hugging Face leaderboard

The [Hugging Face leaderboard](#) provides up-to-date rankings of embedding models. Use it to identify top-performing models for your use case.

When you review models, consider:

- **Tokens:** The size of the model's vocabulary.
- **Memory:** The model's size and inference cost.
- **Dimensions:** The size of the output vectors.

❗ Note

Larger models aren't always better. They might increase cost without improving performance. Use dimensionality reduction and retrieval evaluation to find the right balance.

Understand embedding economics

When you choose an embedding model, you must find a trade-off between performance and cost. Large embedding models usually have better performance on benchmarking datasets.

But, the increased performance adds cost. Large vectors require more space in a vector database. They also require more computational resources and time to compare embeddings. Small embedding models usually have lower performance on the same benchmarks. They require less space in your vector database and less compute and time to compare embeddings.

When you design your system, consider the cost of embedding in terms of storage, compute, and performance requirements. You must validate the performance of the models through experimentation. The publicly available benchmarks are often academic datasets and might not directly apply to your business data and use cases. Depending on the requirements, you can favor performance over cost or accept a trade-off of good-enough performance for lower cost.

Next step

Information-retrieval phase

Related resources

- [Understand embeddings in Azure OpenAI](#)
- [Tutorial: Explore Azure OpenAI embeddings and document search](#)

Last updated on 11/21/2025

Information retrieval

In the previous step of your Retrieval-Augmented Generation (RAG) solution, you generated the embeddings for your chunks. In this step, you generate the index in the vector database and experiment to determine your optimal searches. This article covers configuration options for a search index, types of searches, and reranking strategies.

This article is part of a series. Read the [introduction](#).

Configure your search index

Note

This section describes specific recommendations for Azure AI Search. If you use a different store, review the appropriate documentation to find the key configurations for that service.

The search index in your store has a column for every field in your data. Search stores generally support [nonvector data types](#), such as string, boolean, integer, single, double, and datetime. They also support collections, such as single-type collections and [vector data types](#). For each column, you must [configure information](#), such as the data type and whether the field is filterable, retrievable, or searchable.

Consider the following [vector search configurations](#) that you can apply to vector fields:

- **Vector search algorithm:** The [vector search algorithm](#) searches for relative matches. AI Search has a brute-force algorithm option, called exhaustive k-nearest neighbors (KNN), that scans the entire vector space. It also has a more performant algorithm option, called Hierarchical Navigable Small World (HNSW), that performs an [approximate nearest neighbor \(ANN\)](#) search.
- **Similarity metric:** The algorithm uses a [similarity metric](#) to calculate nearness. The types of metrics in AI Search include cosine, dot product, and Euclidean. If you use Azure OpenAI Service embedding models, choose cosine.
- **The `efConstruction` parameter:** This parameter is used during the construction of an HNSW index. It determines the number of nearest neighbors that are connected to a vector during indexing. A larger `efConstruction` value results in a better-quality index than a smaller number. But a larger value requires more time, storage, and compute. For a large number of chunks, set the `efConstruction` value higher. For a low number of

chunks, set the value lower. To determine the optimal value, experiment with your data and expected queries.

- The `efSearch` parameter: This parameter is used during query time to set the number of nearest neighbors, or similar chunks, that the search uses.
- The `m` parameter: This parameter is the bidirectional link count. The range is 4 to 10. Lower numbers return less noise in the results.

In AI Search, the vector configurations are encapsulated in a `vectorSearch` configuration. When you configure your vector columns, you reference the appropriate configuration for that vector column and set the number of dimensions. The vector column's `dimensions` attribute represents the number of dimensions that your embedding model generates. For example, the storage-optimized *text-embedding-3-small* model generates 1,536 dimensions.

Choose your search approach

When you run queries from your prompt orchestrator against your search store, consider the following factors:

- The type of search that you want to perform, such as vector, keyword, or hybrid
- Whether you want to query against one or more columns
- Whether you want to manually run multiple queries, such as a keyword query and a vector search
- Whether you need to break down the query into subqueries
- Whether you should use filtering in your queries

Your prompt orchestrator might use a static approach or a dynamic approach that combines approaches based on context clues from the prompt. The following sections address these options to help you find the right approach for your workload.

Search types

Search platforms generally support full-text and vector searches. Some platforms, like AI Search, support hybrid searches.

Vector search

[Vector searches](#) compare the similarity between the vectorized query (prompt) and vector fields. For more information, see [Choose an Azure service for vector searches](#).

Important

Before you embed the query, you should perform the same [cleaning operations](#) that you performed on chunks. For example, if you lowercased every word in your embedded chunk, you should lowercase every word in the query before embedding.

Note

You can perform a vector search against multiple vector fields in the same query. In AI Search, this practice is considered a hybrid search. For more information, see [Hybrid search](#).

The following sample code performs a vector search against the contentVector field.

Python

```
embedding = embedding_model.generate_embedding(  
    chunk=str(pre_process.preprocess(query))  
)  
  
vector = RawVectorQuery(  
    k=retrieve_num_of_documents,  
    fields="contentVector",  
    vector=embedding,  
)  
  
results = client.search(  
    search_text=None,  
    vector_queries=[vector],  
    top=retrieve_num_of_documents,  
    select=["title", "content", "summary"],  
)
```

The code that embeds the query preprocesses the query first. That preprocess should be the same code that preprocesses the chunks before embedding. You must use the same embedding model that embedded the chunks.

Full-text search

[Full-text searches](#) match plain text that's stored in an index. It's common practice to extract keywords from a query and use those extracted keywords in a full-text search against one or

more indexed columns. You can configure full-text searches to return matches if any terms or all terms match.

Experiment to determine which fields to run full-text searches against. As described in the [enrichment phase article](#), you should use keyword and entity metadata fields for full-text searches in scenarios where content has similar semantic meaning but entities or keywords differ. Other common fields to consider for full-text search include title, summary, and chunk text.

The following sample code performs a full-text search against the title, content, and summary fields.

Python

```
formatted_search_results = []

results = client.search(
    search_text=query,
    top=retrieve_num_of_documents,
    select=["title", "content", "summary"],
)
formatted_search_results = format_results(results)
```

Hybrid search

AI Search supports [hybrid queries](#) that contain one or more text searches and one or more vector searches. The platform performs each query, gets the intermediate results, reranks the results by using [Reciprocal Rank Fusion](#), and returns the top N results.

The following sample code performs a full-text search against the title, content, and summary fields. It also performs vector searches against the contentVector and questionVector fields. AI Search runs all the queries in parallel, reranks the results, and returns the top $retrieve_num_of_documents$.

Python

```
embedding = embedding_model.generate_embedding(
    chunk=str(pre_process.preprocess(query))
)
vector1 = RawVectorQuery(
    k=retrieve_num_of_documents,
    fields="contentVector",
    vector=embedding,
)
vector2 = RawVectorQuery(
    k=retrieve_num_of_documents,
```

```
        fields="questionVector",
        vector=embedding,
    )

results = client.search(
    search_text=query,
    vector_queries=[vector1, vector2],
    top=retrieve_num_of_documents,
    select=["title", "content", "summary"],
)
```

Manual multiple queries

You can run multiple queries, such as a vector search and a keyword full-text search, manually. You aggregate the results, [rerank](#) the results manually, and return the top results. Consider the following use cases for manual multiple queries:

- You use a search platform that doesn't support hybrid searches. You use manual multiple queries to perform your own hybrid search.
- You want to run full-text searches against different queries. For example, you might extract keywords from the query and run a full-text search against your keywords metadata field. You might then extract entities and run a query against the entities metadata field.
- You want to control the reranking process.
- The query requires that you run [decomposed subqueries](#) to retrieve grounding data from multiple sources.

Query translation

Query translation is an optional step in the information retrieval phase of a RAG solution. This step transforms or translates a query into an optimized form to retrieve better results. Query translation methods include augmentation, decomposition, rewriting, and Hypothetical Document Embeddings (HyDE).

Query augmentation

Query augmentation is a translation step that makes the query simpler and more usable and enhances the context. You should consider augmentation if your query is small or vague. For example, consider the query "Compare the earnings of Microsoft." That query doesn't include time frames or time units to compare and only specifies earnings. Consider an augmented

version of the query, such as "Compare the earnings and revenue of Microsoft in the current year versus last year by quarter." The new query is clear and specific.

When you augment a query, you maintain the original query but add more context. Don't remove or alter the original query, and don't change the nature of the query.

You can use a language model to augment a query. But you can't augment all queries. If you have context, you can pass it along to your language model to augment the query. If you don't have context, you have to determine whether your language model has information that you can use to augment the query. For example, if you use a large language model, like a GPT model, you can determine whether information about the query is readily available on the internet. If so, you can use the model to augment the query. Otherwise, you shouldn't augment the query.

In the following prompt, a language model augments a query. This prompt includes examples for when the query has context and doesn't. For more information, see [RAG experiment accelerator GitHub repository](#).

text

Input Processing:

Analyze the input query to identify the core concept or topic.

Check whether the query provides context.

If context is provided, use it as the primary basis for augmentation and explanation.

If no context is provided, determine the likely domain or field, such as science, technology, history, or arts, based on the query.

Query Augmentation:

If context is provided:

Use the given context to frame the query more specifically.

Identify other aspects of the topic not covered in the provided context that enrich the explanation.

If no context is provided, expand the original query by adding the following elements, as applicable:

Include definitions about every word, such as adjective or noun, and the meaning of each keyword, concept, and phrase including synonyms and antonyms.

Include historical context or background information, if relevant.

Identify key components or subtopics within the main concept.

Request information about practical applications or real-world relevance.

Ask for comparisons with related concepts or alternatives, if applicable.

Inquire about current developments or future prospects in the field.

Other Guidelines:

Prioritize information from provided context when available.
Adapt your language to suit the complexity of the topic, but aim for clarity.
Define technical terms or jargon when they're first introduced.
Use examples to illustrate complex ideas when appropriate.
If the topic is evolving, mention that your information might not reflect the very latest developments.
For scientific or technical topics, briefly mention the level of scientific consensus if relevant.
Use Markdown formatting for better readability when appropriate.

Example Input-Output:

Example 1 (With provided context):

Input: "Explain the impact of the Gutenberg Press"
Context Provided: "The query is part of a discussion about revolutionary inventions in medieval Europe and their long-term effects on society and culture."
Augmented Query: "Explain the impact of the Gutenberg Press in the context of revolutionary inventions in medieval Europe. Cover its role in the spread of information, its effects on literacy and education, its influence on the Reformation, and its long-term impact on European society and culture. Compare it to other medieval inventions in terms of societal influence."

Example 2 (Without provided context):

Input: "Explain CRISPR technology"
Augmented Query: "Explain CRISPR technology in the context of genetic engineering and its potential applications in medicine and biotechnology. Cover its discovery, how it works at a molecular level, its current uses in research and therapy, ethical considerations surrounding its use, and potential future developments in the field."
Now, provide a comprehensive explanation based on the appropriate augmented query.

Context: {context}

Query: {query}

Augmented Query:

Decomposition

Complex queries require more than one collection of data to ground the model. For example, the query "How do electric cars work, and how do they compare to internal combustion engine (ICE) vehicles?" likely requires grounding data from multiple sources. One source might describe how electric cars work, while another compares them to ICE vehicles.

Decomposition is the process of breaking down a complex query into multiple smaller and simpler subqueries. You run each of the decomposed queries independently and aggregate the top results of all the decomposed queries as accumulated context. You then run the original query, which passes the accumulated context to the language model.

You should determine whether the query requires multiple searches before you run any searches. If you require multiple subqueries, you can run [manual multiple queries](#) for all the queries. Use a language model to determine whether multiple subqueries are recommended.

The following prompt categorizes a query as simple or complex. For more information, see [RAG experiment accelerator GitHub repository](#).

text

Consider the given question to analyze and determine whether it falls into one of these categories:

1. Simple, factual question

- a. The question asks for a straightforward fact or piece of information.
- b. The answer can likely be found stated directly in a single passage of a relevant document.

c. Breaking the question down further is unlikely to be beneficial.

Examples: "What year did World War 2 end?", "What is the capital of France?", "What are the features of productX?"

2. Complex, multipart question

a. The question has multiple distinct components or asks for information about several related topics.

b. Different parts of the question likely need to be answered by separate passages or documents.

c. Breaking the question down into subquestions for each component provides better results.

d. The question is open-ended and likely to have a complex or nuanced answer.

e. Answering the question might require synthesizing information from multiple sources.

f. The question might not have a single definitive answer and could warrant analysis from multiple angles.

Examples: "What were the key causes, major battles, and outcomes of the American Revolutionary War?", "How do electric cars work and how do they compare to gas-powered vehicles?"

Based on this rubric, does the given question fall under category 1 (simple) or category 2 (complex)? The output should be in strict JSON format. Ensure that the generated JSON is 100% structurally correct, with proper nesting, comma placement, and quotation marks. There shouldn't be a comma after the last element in the JSON.

Example output:

```
{  
  "category": "simple"  
}
```

You can also use a language model to decompose a complex query. The following prompt decomposes a complex query. For more information, see [RAG experiment accelerator GitHub repository](#).

text

Analyze the following query:

For each query, follow these specific instructions:

- Expand the query to be clear, complete, fully qualified, and concise.
- Identify the main elements of the sentence, typically a subject, an action or relationship, and an object or complement. Determine which element is being asked about or emphasized (usually the unknown or focus of the question). Invert the sentence structure. Make the original object or complement the new subject. Transform the original subject into a descriptor or qualifier. Adjust the verb or relationship to fit the new structure.
- Break the query down into a set of subqueries that have clear, complete, fully qualified, concise, and self-contained propositions.
- Include another subquery by using one more rule: Identify the main subject and object. Swap their positions in the sentence. Adjust the wording to make the new sentence grammatically correct and meaningful. Ensure that the new sentence asks about the original subject.
- Express each idea or fact as a standalone statement that can be understood with the help of the given context.
- Break down the query into ordered subquestions, from least to most dependent.
- The most independent subquestion doesn't require or depend on the answer to any other subquestion or prior knowledge.
- Try having a complete subquestion that has all information only from the base query. There's no other context or information available.
- Separate complex ideas into multiple simpler propositions when appropriate.
- Decontextualize each proposition by adding necessary modifiers to nouns or entire sentences. Replace pronouns, such as it, he, she, they, this, and that, with the full name of the entities that they refer to.
- If you still need more questions, the subquestion isn't relevant and should be removed.

Provide your analysis in the following YAML format, and strictly adhere to the following structure. Don't output anything extra, including the language itself.

```
type: interdependent
queries:
- [First query or subquery]
- [Second query or subquery, if applicable]
- [Third query or subquery, if applicable]
- ...
```

Examples:

1. Query: "What is the capital of France?"

```
type: interdependent
queries:
- What is the capital of France?
```

2. Query: "Who is the current CEO of the company that created the iPhone?"

```
type: interdependent
queries:
- Which company created the iPhone?
- Who is the current CEO of Apple? (identified in the previous question)
```

3. Query: "What is the population of New York City, and what is the tallest building in Tokyo?"

type: multiple_independent

queries:

- What is the population of New York City?
- What is the tallest building in Tokyo?

Now, analyze the following query:

{query}

Rewriting

An input query might not be in the optimal form to retrieve grounding data. You can use a language model to rewrite the query and achieve better results. Rewrite a query to address the following challenges:

- Vagueness
- Missing keywords
- Unnecessary words
- Unclear semantics

The following prompt uses a language model to rewrite a query. For more information, see [RAG experiment accelerator GitHub repository](#).

text

Rewrite the given query to optimize it for both keyword-based and semantic-similarity search methods. Follow these guidelines:

- Identify the core concepts and intent of the original query.
- Expand the query by including relevant synonyms, related terms, and alternate phrasings.
- Maintain the original meaning and intent of the query.
- Include specific keywords that are likely to appear in relevant documents.
- Incorporate natural language phrasing to capture semantic meaning.
- Include domain-specific terminology if applicable to the query's context.
- Ensure that the rewritten query covers both broad and specific aspects of the topic.
- Remove ambiguous or unnecessary words that might confuse the search.
- Combine all elements into a single, coherent paragraph that flows naturally.
- Aim for a balance between keyword richness and semantic clarity.

Provide the rewritten query as a single paragraph that incorporates various search aspects, such as keyword-focused, semantically focused, or domain-specific aspects.

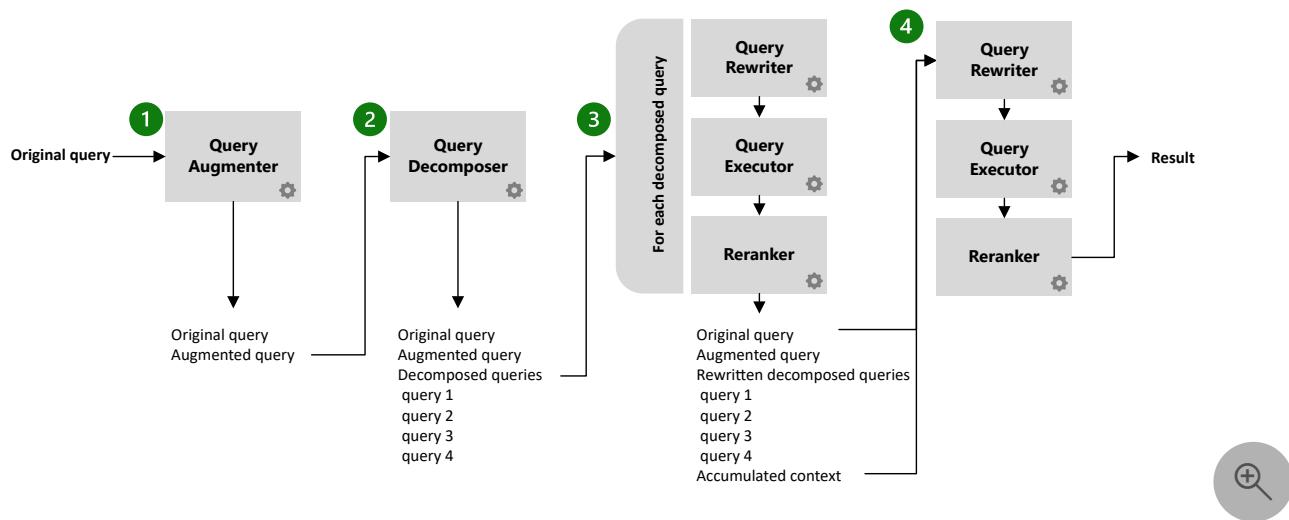
query: {original_query}

The HyDE technique

HyDE[↗] is an alternate information-retrieval technique for RAG solutions. Rather than converting a query into embeddings and using those embeddings to find the closest matches in a vector database, HyDE uses a language model to generate answers from the query. These answers are converted into embeddings, which are used to find the closest matches. This process enables HyDE to perform answer-to-answer embedding-similarity searches.

Combine query translations into a pipeline

You can use multiple query translations. You can even use all four of these translations in conjunction. The following diagram shows an example of how you can combine these translations into a pipeline.



The pipeline has the following steps:

1. The optional query augmenter step receives the original query. This step outputs the original query and the augmented query.
2. The optional query decomposer step receives the augmented query. This step outputs the original query, the augmented query, and the decomposed queries.
3. Each decomposed query performs three substeps. After all the decomposed queries go through the substeps, the output includes the original query, the augmented query, the decomposed queries, and an accumulated context. The accumulated context includes the aggregation of the top N results from all the decomposed queries that go through the substeps. The substeps include the following tasks:
 - a. The optional query rewriter rewrites the decomposed query.
 - b. The search index processes the rewritten query or the original query. It runs the query by using search types, such as vector, full text, hybrid, or manual multiple. The search

index can also use advanced query capabilities, such as HyDE.

- c. The results are reranked. The top N reranked results are added to the accumulated context.
4. The original query, along with the accumulated context, goes through the same three substeps as each decomposed query. But only one query goes through the steps, and the caller receives the top N results.

Pass images in queries

Some multimodal models, such as GPT-4V and GPT-4o, can interpret images. If you use these models, you can avoid chunking your images and pass the image as part of the prompt to the multimodal model. You should experiment to determine how this approach performs compared to chunking the images with and without passing extra context. You should also compare the cost difference and do a cost-benefit analysis.

Filter queries

To filter queries, you can use fields in the search store that are configured as filterable. Consider filtering keywords and entities for queries that use those fields to help narrow down the result. Use filtering to eliminate irrelevant data. Retrieve only the data that satisfies specific conditions from an index. This practice improves the overall performance of the query and provides more relevant results. To determine whether filtering benefits your scenario, do experiments and tests. Consider factors such as queries that don't have keywords or have inaccurate keywords, abbreviations, or acronyms.

Weight fields

In AI Search, you can weight fields to influence the ranking of results based on criteria.

Note

This section describes AI Search weighting capabilities. If you use a different data platform, research the weighting capabilities of that platform.

AI Search supports scoring profiles that contain [parameters for weighted fields and functions for numeric data](#). Scoring profiles only apply to nonvector fields. Support for vector and hybrid search is in preview. You can create multiple scoring profiles on an index and optionally choose to use one on a per-query basis.

The fields that you weight depend on the type of query and the use case. For example, if the query is keyword-centric, such as "Where is Microsoft headquartered?", you want a scoring profile that weights entity or keyword fields higher. You might use different profiles for different users, allow users to choose their focus, or choose profiles based on the application.

In production systems, you should only maintain profiles that you actively use in production.

Use reranking

Use reranking to run one or more queries, aggregate the results, and rank those results. Consider the following scenarios that benefit from reranking search results:

- You performed [manual multiple searches](#), and you want to aggregate the results and rank them.
- Vector and keyword searches aren't always accurate. You want to increase the count of documents that you return from your search, which can include valid results that might otherwise be ignored, and use reranking to evaluate the results.

You can use a language model or cross encoder to perform reranking. Some platforms, like AI Search, have proprietary methods to rerank results. You can evaluate these options for your data to determine what works best for your scenario. The following sections provide details about these methods.

Language model reranking

The following sample language model prompt reranks results. For more information, see [RAG experiment accelerator](#).

text

Each document in the following list has a number next to it along with a summary of the document. A question is also provided.

Respond with the numbers of the documents that you should consult to answer the question, in order of relevance, and the relevance score as a JSON string based on JSON format as shown in the schema section. The relevance score is a number from 1 to 10 based on how relevant you think the document is to the question. The relevance score can be repetitive. Don't output any other text, explanation, or metadata apart from the JSON string. Just output the JSON string, and strip every other text. Strictly remove the last comma from the nested JSON elements if it's present. Don't include any documents that aren't relevant to the question. There should be exactly one document element.

Example format:

Document 1:

content of document 1

```
Document 2:  
content of document 2  
Document 3:  
content of document 3  
Document 4:  
content of document 4  
Document 5:  
content of document 5  
Document 6:  
content of document 6  
Question: user-defined question
```

```
schema:  
{  
    "documents": {  
        "document_1": "Relevance",  
        "document_2": "Relevance"  
    }  
}
```

Cross-encoder reranking

The following example uses a [cross encoder provided by Hugging Face](#) to load the Roberta model. It iterates over each chunk and uses the model to calculate similarity, which provides a value. It sorts the results and returns the top N results. For more information, see [RAG experiment accelerator GitHub repository](#).

Python

```
from sentence_transformers import CrossEncoder  
...  
  
model_name = 'cross-encoder/stsb-roberta-base'  
model = CrossEncoder(model_name)  
  
cross_scores_ques = model.predict(  
    [[user_prompt, item] for item in documents],  
    apply_softmax=True,  
    convert_to_numpy=True,  
)  
  
top_indices_ques = cross_scores_ques.argsort()[-k:][::-1]  
sub_context = []  
for idx in list(top_indices_ques):  
    sub_context.append(documents[idx])
```

Semantic ranking

AI Search has a proprietary feature called [semantic ranking](#). This feature uses deep learning models that were adapted from Microsoft Bing that promote the most semantically relevant results. For more information, see [How semantic ranker works](#).

Consider other search guidance

Consider the following general guidance when you implement your search solution:

- Return the title, summary, source, and the raw uncleaned content fields from a search.
- Determine up front whether you need to break down a query into subqueries.
- Run vector and text queries on multiple fields. When you receive a query, you don't know whether vector search or text search is better. And you don't know the ideal fields that the vector search or keyword search should search. You can search on multiple fields, potentially with multiple queries, rerank the results, and return the results that have the highest scores.
- Filter on keyword and entity fields to narrow down results.
- Use keywords along with vector searches. The keywords filter the results to a smaller subset. The vector store works against that subset to find the best matches.

Evaluate your search results

In the preparation phase, you [gathered test queries along with test document information](#). You can use the following information that you gathered in that phase to evaluate your search results:

- The query: The sample query
- The context: The collection of all the text in the test documents that address the sample query

To evaluate your search solution, you can use the following well-established retrieval evaluation methods:

- **Precision at K:** The percentage of correctly identified relevant items out of the total search results. This metric focuses on the accuracy of your search results.
- **Recall at K:** The percentage of relevant items in the top K out of the total possible relative items. This metric focuses on search results coverage.
- **Mean Reciprocal Rank (MRR):** The average of the reciprocal ranks of the first relevant answer in your ranked search results. This metric focuses on where the first relevant result

occurs in the search results.

You should test positive and negative examples. For the positive examples, you want the metrics to be as close to 1 as possible. For the negative examples, where your data shouldn't be able to address the queries, you want the metrics to be as close to 0 as possible. You should test all your test queries. Average the positive query results and the negative query results to understand how your search results perform in aggregate.

Next step

[LLM end-to-end evaluation phase](#)

Related resources

- [Quickstart: Chat with Azure OpenAI models by using your own data](#)
- [Hybrid search via vectors and full text in AI Search](#)

Last updated on 10/10/2025

Large language model end-to-end evaluation

In this phase, you evaluate your retrieval-augmented generation (RAG) solution by examining the expected user prompts that contain the retrieved grounding data against the language model. Before you reach this phase, you should complete the preceding phases. You must collect your test documents and queries, chunk your test documents, enrich the chunks, embed the chunks, create a search index, and implement a search strategy. Then you should evaluate each of these phases and ensure that the results meet your expectations. At this point, you should be confident that your solution returns relevant grounding data for a user query.

This grounding data forms the context for the prompt that you send to the language model to address the user's query. [Prompt engineering strategies](#) are beyond the scope of this article. This article addresses the evaluation of the engineered call to the language model from the perspective of the grounding data. This article covers common language model evaluation metrics and specific similarity and evaluation metrics that you can use in model evaluation calculations or as standalone metrics.

This article doesn't attempt to provide an exhaustive list of language model metrics or similarity and evaluation metrics. The key takeaway is that different metrics serve specific use cases. Only you have a holistic understanding of your workload. You and your data scientists must determine what you want to measure and which metrics are the most appropriate.

This article is part of a series. Read the [introduction](#) first.

Language model evaluation metrics

There are several metrics that you should use to evaluate the language model's response, including groundedness, completeness, utilization, relevancy, and correctness. Because the overall goal of the RAG pattern is to provide relevant data as context to a language model when generating a response, each of these metrics should ideally score high. But depending on your workload, you might need to prioritize one over another.

Important

Language model responses are nondeterministic, which means that the same prompt to a language model often returns different results. This concept is important to understand when you use a language model as part of your evaluation process. Consider using a target range instead of a single target when you evaluate language model use.

Understand groundedness

Groundedness, sometimes referred to as *faithfulness*, measures whether the response is based completely on the context. It validates that the response isn't using information other than what exists in the context. A low groundedness metric indicates that the language model might be outputting inaccurate or nonsensical responses.

Calculate groundedness

Use the following methods to calculate the groundedness of responses:

- [Azure AI Content Safety-based groundedness](#) is a custom model that uses natural language inference to determine whether claims, or in this case chunks, are based on context in the source document.
- [Large language model-based groundedness](#) uses a language model to determine the level of groundedness of the response.
- [Ragas faithfulness library ↗](#).
- [MLflow faithfulness calculation ↗](#).

Evaluate groundedness

A low groundedness calculation indicates that the language model doesn't see the chunks as relevant. You should evaluate whether you need to add data to your collection, adjust your chunking strategy or chunk size, or fine-tune your prompt.

Understand completeness

Completeness measures whether the response answers all parts of the query. Completeness helps you understand whether the chunks in the context are pertinent, directly relate to the query, and provide a complete answer.

Calculate completeness

You can use the following methods to calculate the completeness of responses:

- Use [AI-assisted retrieval score prompting](#).
- Use a language model to help measure the quality of the language model response. You need the question, context, and generated answer to take this measurement. The following steps outline the high-level process:

1. Use the language model to rephrase, summarize, or simplify the question. This step identifies the intent.
2. Ask the model to check whether the intent or the answer to the intent is found in or can be derived from the retrieved documents. The answer can be "yes" or "no" for each document. Answers that start with "yes" indicate that the retrieved documents are relevant to the intent or answer to the intent.
3. Calculate the ratio of the intents that have an answer that begins with "yes."
4. Square the score to highlight the errors.

Evaluate completeness

If completeness is low, start working to increase it by evaluating your embedding model.

Compare the vocabulary in your content to the vocabulary in your embedding model.

Determine whether you need a domain-specific embedding model or whether you should fine-tune an existing model. The next step is to evaluate your chunking strategy. If you use fixed-sized chunking, consider increasing your chunk size. You can also evaluate whether your test data has enough data to completely address the question.

Understand utilization

Utilization measures the extent to which the response consists of information from the chunks in the context. The goal is to determine the extent to which each chunk is part of the response. Low utilization indicates that your results might not be relevant to the query. You should evaluate utilization alongside completeness.

Calculate utilization

Use a language model to calculate utilization. You can pass the response and the context that contains the chunks to the language model. You can ask the language model to determine the number of chunks that contain the answer.

Evaluate utilization

The following table provides guidance for how to evaluate completeness and utilization.

 Expand table

| | High utilization | Low utilization |
|-------------------|-------------------|---|
| High completeness | No action needed. | The returned data addresses the question but also returns irrelevant chunks. Consider |

| | High utilization | Low utilization |
|-------------------------|--|---|
| | | reducing the top-k parameter value to yield more probable or deterministic results. |
| Low completeness | <p>The language model uses the chunks that you provide, but they don't fully address the question. Consider taking the following steps:</p> <ul style="list-style-type: none"> • Review your chunking strategy to increase the context within the chunks. • Increase the number of chunks by increasing the top-k parameter value. • Evaluate whether you have chunks that weren't returned that can increase the completeness. If so, investigate why they weren't returned. • Follow the guidance in the completeness section. | <p>The returned data doesn't fully answer the question, and the chunks you provide aren't utilized completely. Consider taking the following steps:</p> <ul style="list-style-type: none"> • Review your chunking strategy to increase the context within the chunks. If you use fixed-size chunking, consider increasing the chunk sizes. • Fine-tune your prompts to improve responses. |

Relevance

Relevance measures the extent to which the language model's response is pertinent and related to the query.

Calculate relevance

You can use the following methods to calculate the relevance of responses:

- [AI-assisted: Relevance in Microsoft Foundry](#)
- [Ragas answer relevancy library](#) ↗
- [MLflow relevance calculation](#) ↗

(!) Note

You can use the [Microsoft Foundry portal](#) ↗ to perform the calculations or use the guidance in this article to calculate relevance yourself.

Evaluate relevance

When relevance is low, do the following tasks:

1. Ensure that the chunks provided to the language model are relevant.
 - Determine whether any relevant, viable chunks aren't returned. If you discover these chunks, evaluate your embedding model.
 - If there aren't viable chunks, look to see whether relevant data exists. If it does, evaluate your chunking strategy.
2. If relevant chunks are returned, evaluate your prompt.

The scores that evaluation methods like [completeness](#) output should yield results that are similar to the relevance score.

Understand correctness

Correctness measures the degree to which the response is accurate and factual.

Calculate correctness

You can use the following methods to evaluate correctness:

- **Language model:** Use a language model to calculate correctness. You can pass the response to the language model, ideally a different language model than the one used to generate the result. You can ask the language model to determine whether the response is factual or not.
- **External trusted source:** Use an external trusted source to validate the correctness of the response. Depending on the API of your trusted source, you can use the trusted source alone, or combine it with a language model.

Evaluate correctness

When correctness is low, do the following tasks:

1. Ensure that the chunks provided to the language model are factually correct and there's no data bias. You might need to correct any problems in the source documents or content.
2. If the chunks are factually correct, evaluate your prompt.
3. Evaluate if there are inherent inaccuracies in the model that need to be overcome with more factual grounding data or fine-tuning.

Understand similarity and evaluation metrics

There are hundreds of similarity and evaluation metrics that you can use in data science. Some algorithms are specific to a domain, such as speech-to-text or language-to-language translation. Each algorithm has a unique strategy for calculating its metric.

Data scientists determine what you want to measure and which metric or combination of metrics you can use to measure it. For example, for language translation, the bilingual evaluation understudy (BLEU) metric checks how many n-grams appear in both the machine translation and human translation to measure similarity based on whether the translations use the same words. Cosine similarity uses embeddings between the machine and human translations to measure semantic similarity. If your goal is to have high semantic similarity and use similar words to the human translation, you want a high BLEU score with high cosine similarity. If you only care about semantic similarity, focus on cosine similarity.

The following list contains a sample of common similarity and evaluation metrics. The listed similarity metrics are described as token based, sequence based, or edit based. These descriptions illustrate which approach the metrics use to calculate similarity. The list also contains three algorithms to evaluate the quality of text translation from one language to another.

- [Longest common substring](#) is a sequence-based algorithm that finds the longest common substring between two strings. The longest common substring percentage takes the longest common substring and divides it by the number of characters of the smaller or larger input string.
- [Longest common subsequence \(LCS\)](#) is a sequence-based algorithm that finds the longest subsequence between two strings. LCS doesn't require the subsequences to be in consecutive order.
- [Cosine similarity](#) is a token-based algorithm that calculates the cosine of the angle between the two vectors.
- [Jaro-Winkler distance](#) is an edit-based algorithm that counts the minimum number of steps to transform one string into another.
- [Hamming distance](#) is an edit-based algorithm that measures the minimum number of substitutions that are required to transform one string into another.
- [Jaccard index](#) is a token-based algorithm that calculates similarity by dividing the intersection of two strings by the union of those strings.
- [Levenshtein distance](#) is an edit-based algorithm that calculates similarity by determining the minimum number of single character edits that are required to transform one string into another.
- [BLEU](#) evaluates the quality of text that's the result of machine translation from one language to another. BLEU calculates the overlap of n-grams between a machine translation and a human-quality translation to make this evaluation.
- [ROUGE](#) is a metric that compares a machine translation of one language to another to a human-created translation. There are several ROUGE variants that use the overlap of n-

grams, skip-bigrams, or longest common subsequence.

- [METEOR](#) evaluates the quality of text that's the result of machine translation by looking at exact matches, matches after stemming, synonyms, paraphrasing, and alignment.

For more information about common similarity and evaluation metrics, see the following resources:

- [PyPi textdistance package](#)
- [Wikipedia list of similarity algorithms](#)

Use multiple evaluation metrics together

You should use the language model evaluation metrics together to get a better understanding of how well your RAG solution performs. The following examples describe how to use multiple evaluation metrics together.

Groundedness and correctness

Groundedness and correctness metrics together help determine if the system is accurately interpreting and using the context. If groundedness is high but correctness is low, it means the language model is using the context but providing an incorrect response. Improper use of context or problems with the source data can cause an incorrect response. For example, if groundedness is 0.9 but correctness is 0.4, it indicates that the system is referencing the correct source material but drawing incorrect conclusions. For example, a response might state *Einstein developed quantum mechanics* based on a context that separately mentions both Einstein and quantum mechanics. This response is grounded but factually incorrect.

This metric combination might require you to prioritize one metric over the other, depending on your specific workload. For example, if the source data contains potentially false information by design, it might be critical for the system to retain that false information in its responses. In that case, you want to prioritize a grounded response over a correct response. In other cases, you might want your workload to consult context data but still make correctness the priority.

Utilization and completeness

Utilization and completeness metrics together help evaluate the effectiveness of the retrieval system. High utilization (0.9) with low completeness (0.3) indicates that the system retrieves accurate but incomplete information. For instance, when a user asks the system about World War II causes, it might retrieve correct information about the invasion of Poland but miss other crucial factors. This scenario might indicate that there are chunks with relevant information that

aren't used as part of the context. To address this scenario, consider returning more chunks, evaluating your chunk ranking strategy, and evaluating your prompt.

Groundedness, utilization, and similarity

Groundedness, utilization, and similarity metrics together help identify how well the system maintains truth while transforming information. High groundedness (0.9) and utilization (.9) with low similarity (0.3) indicates that the system is using accurate grounding data, but paraphrasing poorly. To address this scenario, evaluate your prompt. Modify the prompt and test the results.

Documentation, reporting, and aggregation

You should document both the hyperparameters that you choose for an experiment and the resulting evaluation metrics so that you can understand how the hyperparameters affect your results. You should document hyperparameters and results at granular levels, like embedding or search evaluation, and at a macro level, like testing the entire system end to end.

During design and development, you might be able to track the hyperparameters and results manually. But performing multiple evaluations against your entire test document and test query collection might involve hundreds of evaluation runs and thousands of results. You should automate the persistence of parameters and results for your evaluations.

After your hyperparameters and results are persisted, you should consider making charts and graphs to help you visualize how the hyperparameters affect the metrics. Visualization helps you identify which choices lead to dips or spikes in performance.

It's important to understand that designing and evaluating your RAG solution isn't a one-time operation. Your collection of documents changes over time. The questions that your customers ask change over time, and your understanding of the types of questions evolves as you learn from production. You should revisit this process again and again. Maintaining documentation of past evaluations is critical for future design and evaluation efforts.

Responsible AI, content safety, and security evaluation

As RAG systems become more deeply integrated into enterprise workflows, evaluation must extend beyond traditional metrics like groundedness and completeness. Responsible AI practices require assessing how retrieved and generated content aligns with safety, privacy,

and ethical standards. This section outlines key dimensions of responsible evaluation that help ensure your RAG solution is effective while staying secure and trustworthy.

Content safety evaluation

RAG systems might retrieve or generate harmful content from knowledge bases, which can pose psychological, social, or physical risks. Your evaluation should include detection and mitigation of the following types of content:

- **Hate speech and bias** that target individuals or groups based on race, ethnicity, nationality, gender, sexual orientation, religion, immigration status, ability, appearance, or body size
- **Violent content**, including depictions of weapons, physical harm, or intent to injure or kill
- **Self-harm references**, such as descriptions of suicide or bodily injury
- **Sexual content**, including explicit anatomical references, erotic acts, sexual violence, pornography, or abuse

Use tools like [Content Safety](#) to flag and score retrieved content for harmful elements. Incorporate these scores into your evaluation pipeline to ensure safe outputs.

Intellectual property protection

You must evaluate RAG systems for inadvertent retrieval or generation of copyrighted material, which includes:

- **Textual works** such as song lyrics, articles, recipes, and proprietary documents.
- **Visual works** including logos, brand assets, artworks, fictional characters, and other protected imagery.

Implement content filters and copyright detection models to identify and exclude protected material from retrieval sources and generated responses.

Security and adversarial threats

RAG systems are vulnerable to indirect prompt injection attacks, where malicious instructions are embedded in retrieved documents. These attacks can result in:

- **Unauthorized data exposure** from the knowledge base.
- **Manipulated responses** because of corrupted documents.

- **Bypassing of safety controls** via compromised retrieval sources.

Security evaluation should include adversarial testing, document sanitization, and monitoring for anomalous retrieval patterns.

Privacy and data protection

Evaluating privacy risks is essential when RAG systems interact with repositories that contain sensitive or personal data. This type of data includes:

- **Direct identifiers:** Names, Social Security numbers, passport numbers, and national ID numbers.
- **Contact information:** Email addresses, phone numbers, physical addresses, and IP addresses.
- **Financial data:** Credit card numbers, bank account details, and transaction records.
- **Biometric data:** Fingerprints, facial recognition, voice prints, and retinal scans.
- **Health information:** Medical records, insurance details, and treatment history, like protected health information (PHI) and Health Insurance Portability and Accountability Act (HIPAA).
- **Employment data:** Employee IDs, salary, and performance reviews.
- **Credentials:** Usernames, passwords, API keys, and access tokens.

Use automated personal data detection tools and enforce strict access controls to prevent privacy violations during retrieval and generation.

Key considerations

- **Source quality is critical:** Curated and vetted documents reduce the risk of harmful or inappropriate content.
- **Retrieval can amplify risks:** Poor retrieval strategies might surface unsafe content more frequently than random chance.
- **Document corruption is a unique threat:** Malicious actors can inject harmful content into knowledge bases.
- **Context windows matter:** Mixing legitimate and malicious content complicates detection and evaluation.

- **Continuous monitoring is required:** Regular audits of document repositories and retrieval patterns are essential to maintain safety and trust.

The RAG Experiment Accelerator repository

The articles in this series walk you through all the phases and design choices that are involved in designing and evaluating a RAG solution. They focus on what you should do, not how to do it. An engineering team that works with Microsoft top customers developed a tool called the [RAG Experiment Accelerator](#). The RAG Experiment Accelerator is a custom, code-based experimentation framework. It optimizes and enhances the development of RAG solutions. The framework empowers researchers and developers to efficiently explore and fine-tune the critical components that drive RAG performance. This innovation ultimately results in more accurate and coherent text generation.

The implementation in the repository uses a command-line interface (CLI), so you can easily experiment with various embedding models, refine chunking strategies, and evaluate different search approaches to unlock the full potential of your RAG system. It helps you focus on the core aspects of RAG development by using a simple configuration for hyperparameter tuning.

The framework also provides comprehensive support for language model configuration. This support helps you strike the perfect balance between model complexity and generation quality. The tool simplifies experimentation, saves time, and improves the performance of your RAG models.

RAG with Vision Application Framework

Much of the guidance in this article about working with media in your RAG solution came from another engineering team that works with Microsoft top customers. The team wrote a framework called the [RAG with Vision Application Framework](#). The framework provides a Python-based RAG pipeline that processes both textual and image content from MHTML documents.

The framework loads, chunks, and enriches text and images from MHTML files. It then ingests the chunks into Azure AI Search. The framework implements caching for image enrichment for processing and cost efficiency. It also incorporates evaluation as part of the pipeline.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

- [Raouf Aliouat](#) | Software Engineer II

- [Rob Bagby](#) | Principal Content Developer - Azure Patterns & Practice
- [Paul Butler](#) | Software Engineer
- [Prabal Deb](#) | Principal Software Engineer
- [Soubhi Hadri](#) | Senior Data Scientist
- [Ritesh Modi](#) | Principal Engineer
- [Ryan Pfalz](#) | Senior Technical Program Manager
- [Mahdi Setayesh](#) | Principal Software Engineer
- [Randy Thurman](#) | Principal AI Cloud Solution Architect

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

[RAG Experiment Accelerator GitHub repository](#)

[RAG with Vision Application Framework GitHub repository](#)

Related resource

- [Develop an evaluation flow](#)

Last updated on 11/21/2025

Design a secure multitenant RAG inferencing solution

10/03/2025

Retrieval-Augmented Generation (RAG) is a pattern for building applications that use foundation models to reason over proprietary information or other data that isn't publicly available on the internet. Generally, a client application calls to an orchestration layer that fetches relevant information from a data store, such as a vector database. The orchestration layer passes that data as part of the context as grounding data to the foundation model.

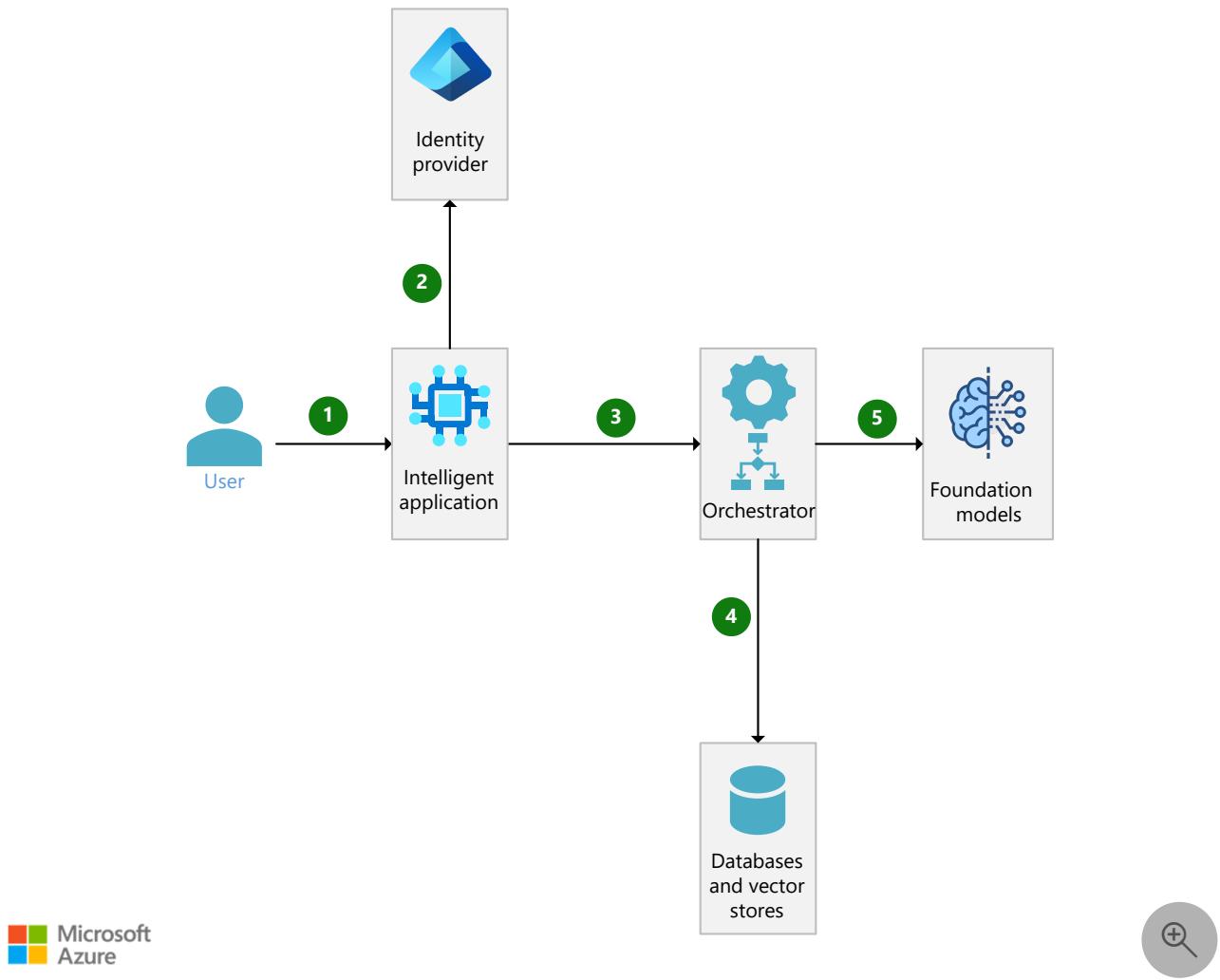
A multitenant solution is used by multiple customers. Each customer, or tenant, consists of multiple users from the same organization, company, or group. In multitenant scenarios, you need to make sure that tenants, or individuals within tenants, are only able to incorporate grounding data that they're authorized to access.

There are multitenant concerns beyond ensuring that users only access the information they're authorized to access. However, this article focuses on that aspect of multitenancy. This article begins with an overview of single-tenant RAG architectures. It discusses the challenges that you might encounter in multitenancy with RAG and some common approaches to take. It also outlines multitenancy considerations and recommendations for improved security.

ⓘ Note

This article describes several features that are specific to Azure OpenAI Service, such as the Azure OpenAI On Your Data feature. However, you can apply most of the principles described in this article to foundational AI models on any platform.

Single-tenant RAG architecture with an orchestrator



Workflow

In this single-tenant RAG architecture, an orchestrator fetches relevant proprietary tenant data from the data stores and provides it as grounding data to the foundation model. The following steps describe a high-level workflow.

1. A user issues a request to the intelligent web application.
2. An identity provider authenticates the requestor.
3. The intelligent application calls the orchestrator API with the user's query and the authorization token for the user.
4. The orchestration logic extracts the user's query from the request and calls the appropriate data store to fetch relevant grounding data for the query. The grounding data is added to the prompt that's sent to the foundation model, like a model that's exposed in Azure OpenAI, in the next step.
5. The orchestration logic connects to the foundation model's inferencing API and sends the prompt that includes the retrieved grounding data. The results are returned to the intelligent application.

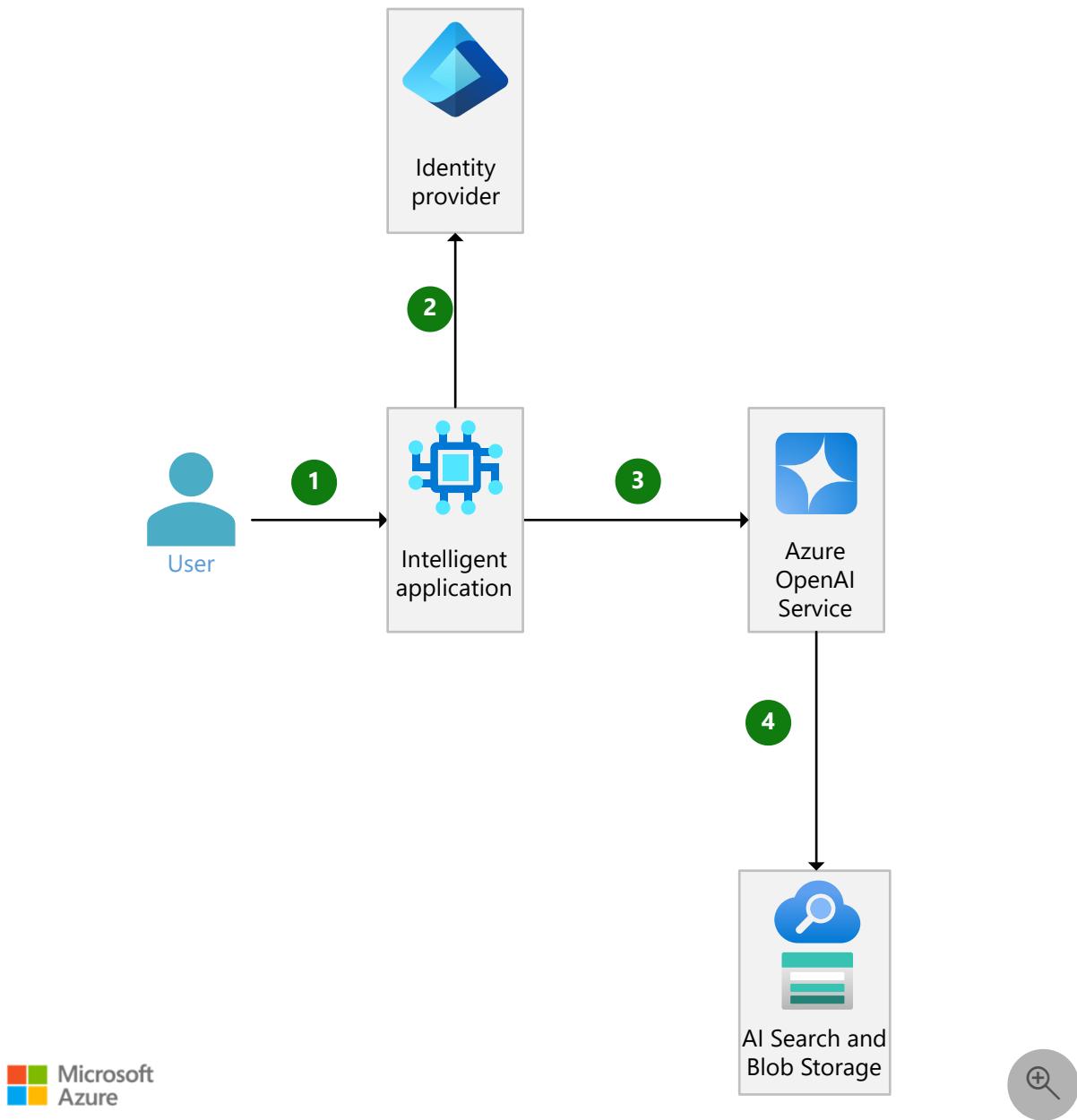
For more information, see [Design and develop a RAG solution](#).

Single-tenant RAG architecture with direct data access

This variant of the single-tenant RAG architecture uses the [On Your Data feature](#) of Azure OpenAI to integrate directly with data stores like Azure AI Search. In this architecture, you either don't have your own orchestrator, or your orchestrator has fewer responsibilities. The Azure OpenAI API calls into the data store to fetch the grounding data and passes that data to the language model. This method gives you less control over what grounding data to fetch and the relevancy of that data.

Note

Azure OpenAI is managed by Microsoft. It integrates with the data store, but the model itself doesn't integrate with the data store. The model receives grounding data in the same way as it does when an orchestrator fetches the data.



Workflow

In this RAG architecture, the service that provides the foundation model fetches the appropriate proprietary tenant data from the data stores and uses that data as grounding data to the foundation model. The following steps describe a high-level workflow. The italicized steps are identical to the preceding single-tenant RAG architecture that has an orchestrator workflow.

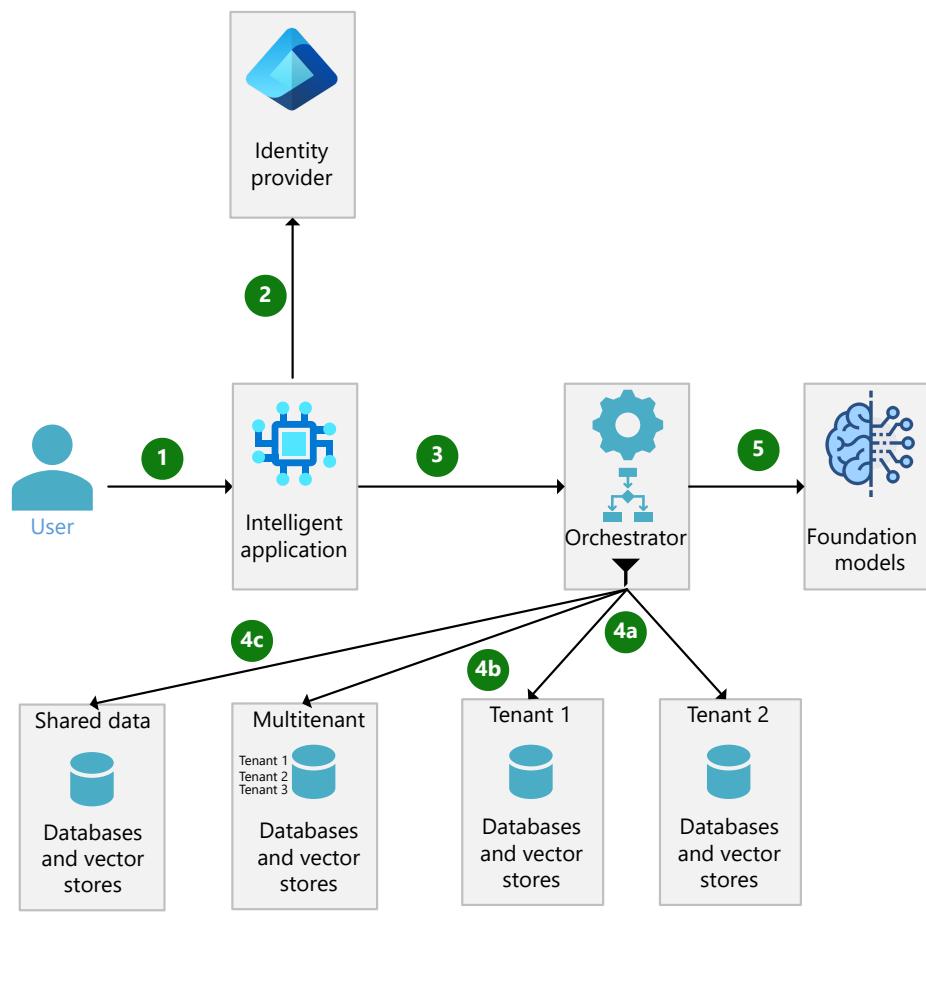
1. *A user issues a request to the intelligent web application.*
2. *An identity provider authenticates the requestor.*
3. The intelligent application calls Azure OpenAI with the user's query.
4. Azure OpenAI connects to supported data stores, such as AI Search and Azure Blob Storage, to fetch the grounding data. The grounding data is used as part of the context

when Azure OpenAI calls the OpenAI language model. The results are returned to the intelligent application.

If you want to use this architecture in a multitenant solution, then the service that directly accesses the grounding data, such as Azure OpenAI, must support the multitenant logic that your solution requires.

Multitenancy in RAG architecture

In multitenant solutions, tenant data might exist in a tenant-specific store or coexist with other tenants in a multitenant store. Data might also be in a store that's shared across tenants. Only data that the user is authorized to access should be used as grounding data. The user should see only common or all-tenant data or data from their tenant that's filtered to help ensure that they see only the data that they're authorized to access.



Workflow

The following steps describe a high-level workflow. The italicized steps are identical to the [single-tenant RAG architecture with an orchestrator](#) workflow.

1. A user issues a request to the intelligent web application.
2. An identity provider authenticates the requestor.
3. The intelligent application calls the orchestrator API with the user's query and the authorization token for the user.
4. The orchestration logic extracts the user's query from the request and calls the appropriate data stores to fetch tenant-authorized, relevant grounding data for the query. The grounding data is added to the prompt that's sent to Azure OpenAI in the next step. Some or all of the following steps are included:
 - a. The orchestration logic fetches grounding data from the appropriate tenant-specific data store instance and potentially applies security filtering rules to return only the data that the user is authorized to access.
 - b. The orchestration logic fetches the appropriate tenant's grounding data from the multitenant data store and potentially applies security filtering rules to return only the data that the user is authorized to access.
 - c. The orchestration logic fetches data from a data store that's shared across tenants.
5. The orchestration logic connects to the foundation model's inferencing API and sends the prompt that includes the retrieved grounding data. The results are returned to the intelligent application.

Design considerations for multitenant data in RAG

Consider the following options when you design your multitenant RAG inferencing solution.

Choose a store isolation model

The two main [architectural approaches for storage and data in multitenant scenarios](#) are store-per-tenant and multitenant stores. These approaches are in addition to stores that contain data shared across tenants. Your multitenant solution can use a combination of these approaches.

Store-per-tenant stores

In store-per-tenant stores, each tenant has its own store. The advantages of this approach include both data and performance isolation. Each tenant's data is encapsulated in its own store. In most data services, the isolated stores aren't susceptible to the noisy neighbor problem of other tenants. This approach also simplifies cost allocation because the entire cost of a store deployment can be attributed to a single tenant.

This approach might present challenges such as increased management and operational overhead and higher costs. You shouldn't use this approach if you have a large number of

small tenants, like in business-to-consumer scenarios. This approach might also reach or exceed [service limits](#).

In the context of this AI scenario, a store-per-tenant store means that the necessary grounding data to bring relevancy into the context comes from an existing or new data store that only contains grounding data for the tenant. In this topology, the database instance is the discriminator that's used for each tenant.

Multitenant stores

In multitenant stores, multiple tenants' data coexists in the same store. The advantages of this approach include the potential for cost optimization, the ability to handle a higher number of tenants than the store-per-tenant model, and lower management overhead because of the lower number of store instances.

The challenges of using shared stores include the need for data isolation and management, the potential for the [noisy neighbor antipattern](#), and more complex cost allocation to tenants. Data isolation is the most important concern when you use this approach. You need to implement secure approaches to help ensure that tenants can only access their data. Data management can also be challenging if tenants have different data lifecycles that require operations such as building indexes on different schedules.

Some platforms have features that you can use when you implement tenant data isolation in shared stores. For example, Azure Cosmos DB has native support for data partitioning and sharding. It's typical to use a tenant identifier as a partition key to provide some isolation between tenants. Azure SQL and Azure Database for PostgreSQL - Flexible Server support row-level security. However, these features aren't typically used in multitenant solutions because you have to design your solution around these features if you plan to use them in your multitenant store.

In the context of this AI scenario, grounding data for all tenants commingle in the same data store. Therefore, your query to that data store must include a tenant discriminator to help ensure that responses are restricted to bring back only relevant data within the context of the tenant.

Shared stores

Multitenant solutions often share data across tenants. In an example multitenant solution for the healthcare domain, a database might store general medical information or information that isn't specific to the tenant.

In the context of this AI scenario, the grounding data store is generally accessible and doesn't need filtering based on specific tenants because the data is relevant and authorized for all tenants in the system.

Identity

Identity is a key aspect of multitenant solutions, including multitenant RAG solutions. The intelligent application should integrate with an identity provider to authenticate the identity of the user. The multitenant RAG solution needs an [identity directory](#) that stores authoritative identities or references to identities. This identity needs to flow through the request chain and allow downstream services, such as the orchestrator or the data store itself, to identify the user.

You also need a way to [map a user to a tenant](#) so that you can grant access to that tenant data.

Define your tenant and authorization requirements

When you build a multitenant RAG solution, you must [define what a tenant is for your solution](#). The two common models to choose from are business-to-business and business-to-consumer models. The model that you choose helps you determine what other factors you should consider when you build your solution. Understanding the number of tenants is critical for choosing the data store model. A large number of tenants might require a model that has multiple tenants for each store. A smaller number of tenants might allow for a store-per-tenant model. The amount of data for each tenant is also important. Tenants that have large amounts of data might prevent you from using multitenant stores because of size limitations on the data store.

If you intend to expand an existing workload to support this AI scenario, you might have made this decision already. Generally speaking, you can use your existing data storage topology for the grounding data if that data store can provide sufficient relevancy and meet any other nonfunctional requirements. However, if you plan to introduce new components, such as a dedicated vector search store as a dedicated grounding store, then you still need to make this decision. Consider factors such as your current deployment stamp strategy, your application control plane impact, and any per-tenant data lifecycle differences, such as pay-for-performance situations.

After you define what a tenant is for your solution, you need to define your authorization requirements for data. Tenants only access data from their tenant, but your authorization requirements might be more granular. For example, in a healthcare solution, you might have rules such as:

- A patient can only access their own patient data.
- A healthcare professional can access their patients' data.

- A finance user can access only finance-related data.
- A clinical auditor can see all patients' data.
- All users can access basic medical knowledge in a shared data store.

In a document-based RAG application, you might want to restrict users' access to documents based on a tagging scheme or sensitivity levels assigned to the documents.

After you have a definition of what a tenant is and have a clear understanding of the authorization rules, use that information as requirements for your data store solution.

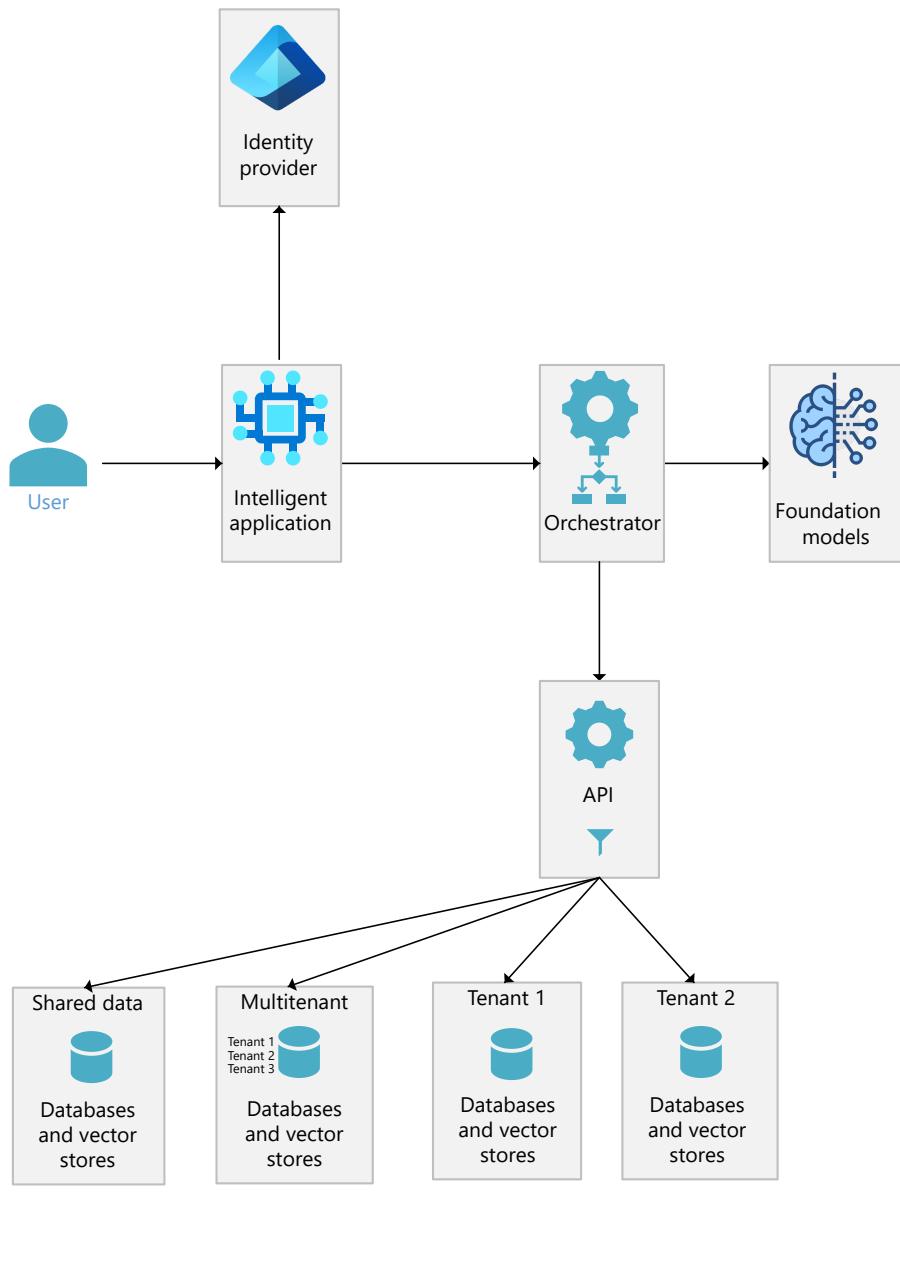
Data filtering

Restricting access to only the data that users are authorized to access is known as *filtering* or *security trimming*. In a multitenant RAG scenario, a user might be mapped to a tenant-specific store. That doesn't mean that the user should be able to access all the data in that store. [Define your tenant and authorization requirements](#) discusses the importance of defining authorization requirements for your data. You should use these authorization rules as the basis for filtering.

You can use data platform capabilities like row-level security to implement filtering. Or you might need custom logic, data, or metadata. These platform features aren't typically used in multitenant solutions because you need to design your system around these features.

Encapsulate multitenant data logic

We recommend that you have an API in front of the storage mechanism that you use. The API acts like a gatekeeper that helps ensure that users only get access to information they're authorized to access.



Users' access to data can be limited by:

- The user's tenant.
- Platform features.
- Custom security filtering or trimming rules.

The API layer should:

- Route the query to a tenant-specific store in a store-per-tenant model.
- Select only data from the user's tenant in multitenant stores.
- Use the appropriate identity for a user to support platform-enabled authorization logic.
- Enforce custom security trimming logic.
- Store access logs of grounding information for audit purposes.

Code that needs to access tenant data shouldn't be able to query the back-end stores directly. All requests for data should flow through the API layer. This API layer provides a single point of governance or security on top of your tenant data. This approach prevents the tenant and user data access authorization logic from reaching other areas of the application. This logic is encapsulated in the API layer. This encapsulation makes the solution easier to validate and test.

Summary

When you design a multitenant RAG inferencing solution, you must consider how to architect the grounding data solution for your tenants. Understand the number of tenants and the amount of per-tenant data that you store. This information helps you design your data tenancy solution. We recommend that you implement an API layer that encapsulates the data access logic, including multitenant logic and filtering logic.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

Principal authors:

- [John Downs](#) | Principal Software Engineer, Azure Patterns & Practices
- [Daniel Scott-Raynsford](#) | Sr. Partner Solution Architect, Data & AI

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next step

[Design and develop a RAG solution](#)

Related resources

- [SaaS and multitenant solution architecture](#)
- [Basic AI Foundry chat architecture](#)
- [Baseline AI Foundry chat reference architecture](#)
- [Access Azure OpenAI and other language models through a gateway](#)

Machine learning operations

Article • 07/12/2024

This article describes three Azure architectures for machine learning operations that have end-to-end continuous integration and continuous delivery (CI/CD) pipelines and retraining pipelines. The architectures are for these AI applications:

- Classical machine learning
- Computer vision (CV)
- Natural language processing

These architectures are the product of the MLOps v2 project. They incorporate best practices that solution architects identified in the process of developing various machine learning solutions. The result is deployable, repeatable, and maintainable patterns. All three architectures use the Azure Machine Learning service.

For an implementation with sample deployment templates for MLOps v2, see [Azure MLOps v2 solution accelerator](#).

Potential use cases

- Classical machine learning: Time-series forecasting, regression, and classification on tabular structured data are the most common use cases in this category. Examples include:
 - Binary and multi-label classification.
 - Linear, polynomial, ridge, lasso, quantile, and Bayesian regression.
 - ARIMA, autoregressive, SARIMA, VAR, SES, LSTM.
- CV: The MLOps framework in this article focuses mostly on the CV use cases of segmentation and image classification.
- Natural language processing: You can use this MLOps framework to implement:
 - Named entity recognition:
 - Text classification
 - Text generation
 - Sentiment analysis
 - Translation

- Question answering
- Summarization
- Sentence detection
- Language detection
- Part-of-speech tagging

AI simulations, deep reinforcement learning, and other forms of AI aren't described in this article.

Architecture

The MLOps v2 architectural pattern has four main modular components, or phases, of the MLOps lifecycle:

- Data estate
- Administration and setup
- Model development, or the inner loop phase
- Model deployment, or the outer loop phase

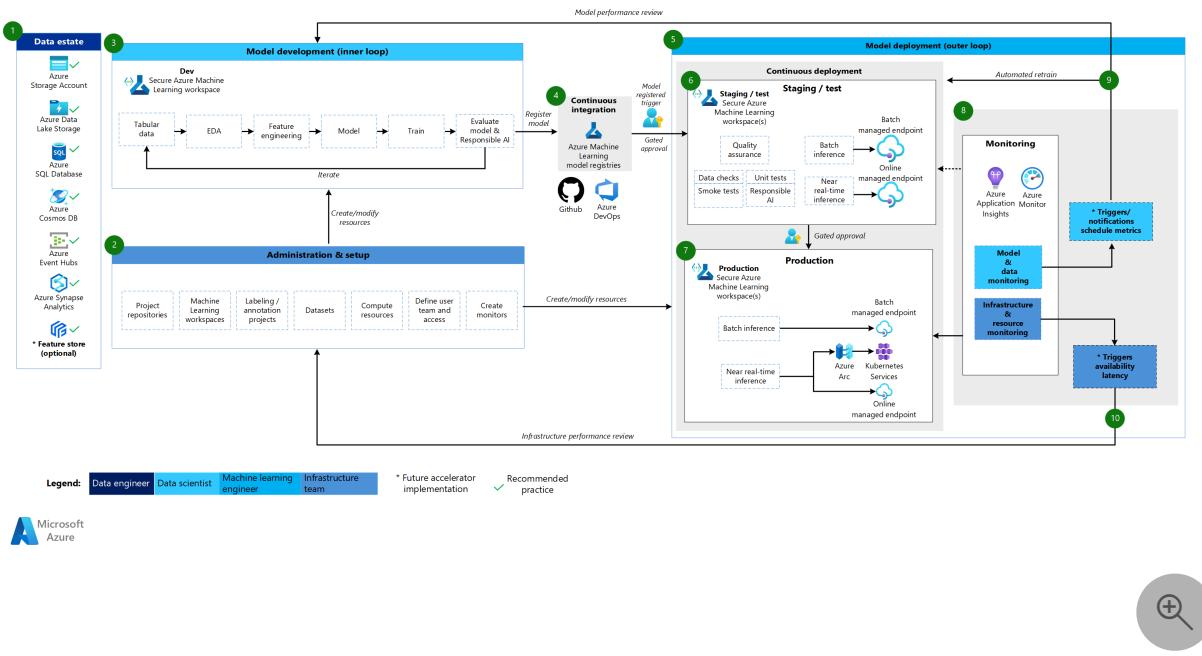
The preceding components, the connections between them, and the typical personas involved are standard across all MLOps v2 scenario architectures. Variations in the details of each component depend on the scenario.

The base architecture for MLOps v2 for Machine Learning is the classical machine learning scenario for tabular data. The CV and NLP architectures build on and modify this base architecture.

MLOps v2 covers the following architectures that are described in this article:

- [Classical machine learning architecture](#)
- [Machine Learning CV architecture](#)
- [Machine Learning natural language processing architecture](#)

Classical machine learning architecture



Download a [Visio file](#) of this architecture.

Workflow for the classical machine learning architecture

1. Data estate

This component illustrates the data estate of the organization and potential data sources and targets for a data science project. Data engineers are the primary owners of this component of the MLOps v2 lifecycle. The Azure data platforms in this diagram aren't exhaustive or prescriptive. A green check mark indicates the data sources and targets that represent recommended best practices that are based on the customer use case.

2. Administration and setup

This component is the first step in the MLOps v2 accelerator deployment. It consists of all tasks related to the creation and management of resources and roles that are associated with the project. For example, the infrastructure team might:

- Create project source code repositories.
- Use Bicep or Terraform to create Machine Learning workspaces.
- Create or modify datasets and compute resources for model development and deployment.
- Define project team users, their roles, and access controls to other resources.
- Create CI/CD pipelines.

- f. Create monitoring components to collect and create alerts for model and infrastructure metrics.

The primary persona associated with this phase is the infrastructure team, but an organization might also have data engineers, machine learning engineers, or data scientists.

3. Model development (inner loop phase)

The inner loop phase consists of an iterative data science workflow that acts within a dedicated and secure Machine Learning workspace. The preceding diagram shows a typical workflow. The process starts with data ingestion, moves through exploratory data analysis, experimentation, model development and evaluation, and then registers a model for production use. This modular component as implemented in the MLOps v2 accelerator is agnostic and adaptable to the process that your data science team uses to develop models.

Personas associated with this phase include data scientists and machine learning engineers.

4. Machine Learning registries

After the data science team develops a model that they can deploy to production, they register the model in the Machine Learning workspace registry. CI pipelines that are triggered, either automatically by model registration or by gated human-in-the-loop approval, promote the model and any other model dependencies to the model deployment phase.

Personas associated with this stage are typically machine learning engineers.

5. Model deployment (outer loop phase)

The model deployment, or outer loop phase, consists of preproduction staging and testing, production deployment, and monitoring of the model, data, and infrastructure. When the model meets the criteria of the organization and use case, CD pipelines promote the model and related assets through production, monitoring, and potential retraining.

Personas associated with this phase are primarily machine learning engineers.

6. Staging and test

The staging and test phase varies according to customer practices. This phase typically includes operations such as retraining and testing the model candidate on production data, test deployments for endpoint performance, data quality checks, unit testing, and responsible AI checks for model and data bias. This phase takes place in one or more dedicated and secure Machine Learning workspaces.

7. Production deployment

After a model passes the staging and test phase, machine learning engineers can use human-in-the-loop gated approval to promote it to production. Model deployment options include a managed batch endpoint for batch scenarios or either a managed online endpoint or Kubernetes deployment that uses Azure Arc for online, near real-time scenarios. Production typically takes place in one or more dedicated and secure Machine Learning workspaces.

8. Monitoring

Machine learning engineers monitor components in staging, testing, and production to collect metrics related to changes in performance of the model, data, and infrastructure. They can use those metrics to take action. Model and data monitoring can include checking for model and data drift, model performance on new data, and responsible AI problems. Infrastructure monitoring might identify slow endpoint response, inadequate compute capacity, or network problems.

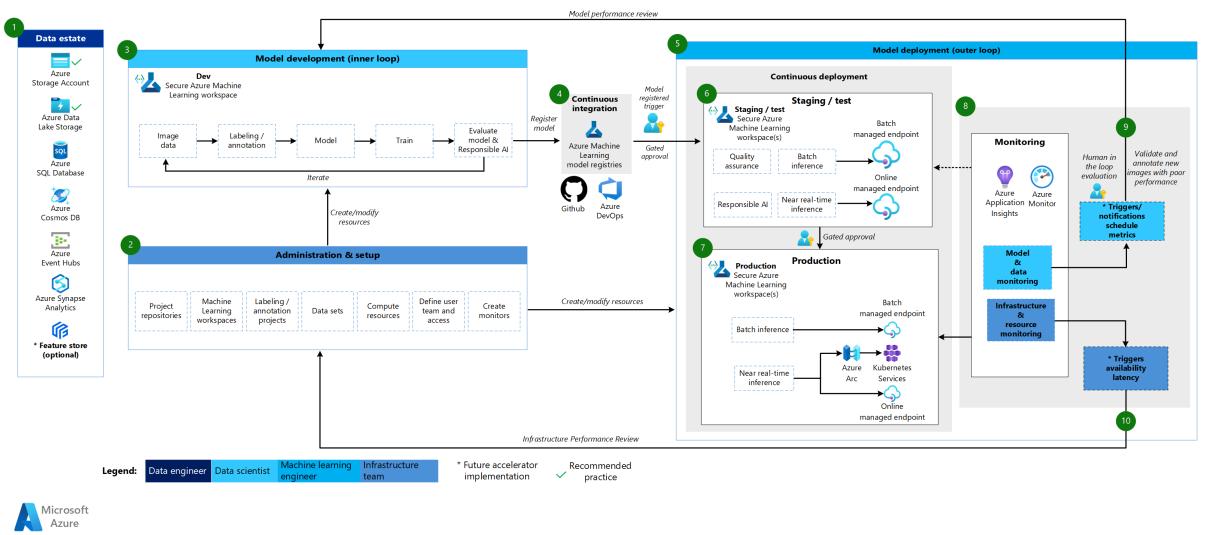
9. Data and model monitoring: events and actions

Based on model and data criteria, such as metric thresholds or schedules, automated triggers and notifications can implement appropriate actions to take. For example, a trigger might retrain a model to use new production data and then loopback the model to staging and testing for a preproduction evaluation. Or a model or data problem might trigger an action that requires a loopback to the model development phase where data scientists can investigate the problem and potentially develop a new model.

10. Infrastructure monitoring: events and actions

Automated triggers and notifications can implement appropriate actions to take based on infrastructure criteria, such as an endpoint response lag or insufficient compute for the deployment. Automatic triggers and notifications might trigger a loopback to the setup and administration phase where the infrastructure team can investigate the problem and potentially reconfigure the compute and network resources.

Machine Learning CV architecture



[Download a Visio file](#) of this architecture.

Workflow for the CV architecture

The Machine Learning CV architecture is based on the classical machine learning architecture, but it has modifications that are specific to supervised CV scenarios.

1. Data estate

This component demonstrates the data estate of the organization and potential data sources and targets for a data science project. Data engineers are the primary owners of this component in the MLOps v2 lifecycle. The Azure data platforms in this diagram aren't exhaustive or prescriptive. Images for CV scenarios can come from various data sources. For efficiency when developing and deploying CV models with Machine Learning, we recommend Azure Blob Storage and Azure Data Lake Storage.

2. Administration and setup

This component is the first step in the MLOps v2 accelerator deployment. It consists of all tasks related to the creation and management of resources and roles associated with the project. For CV scenarios, administration and setup of the MLOps v2 environment is largely the same as for classical machine learning but includes an extra step. The infrastructure team uses the labeling feature of Machine Learning or another tool to create image labeling and annotation projects.

3. Model development (inner loop phase)

The inner loop phase consists of an iterative data science workflow performed within a dedicated and secure Machine Learning workspace. The primary difference between this workflow and the classical machine learning scenario is that image labeling and annotation is a key component of this development loop.

4. Machine Learning registries

After the data science team develops a model that they can deploy to production, they register the model in the Machine Learning workspace registry. CI pipelines that are triggered automatically by model registration or by gated human-in-the-loop approval promote the model and any other model dependencies to the model deployment phase.

5. Model deployment (outer loop phase)

The model deployment or outer loop phase consists of preproduction staging and testing, production deployment, and monitoring of the model, data, and infrastructure. When the model meets the criteria of the organization and use case, CD pipelines promote the model and related assets through production, monitoring, and potential retraining.

6. Staging and test

The staging and test phase varies according to customer practices. This phase typically includes operations such as test deployments for endpoint performance, data quality checks, unit testing, and responsible AI checks for model and data bias. For CV scenarios, machine learning engineers don't need to retrain the model candidate on production data because of resource and time constraints. The data science team can instead use production data for model development. The candidate model registered from the development loop is evaluated for production. This phase takes place in one or more dedicated and secure Machine Learning workspaces.

7. Production deployment

After a model passes the staging and test phase, machine learning engineers can use human-in-the-loop gated approval to promote it to production. Model deployment options include a managed batch endpoint for batch scenarios or either a managed online endpoint or Kubernetes deployment that uses Azure Arc for online, near real-time scenarios. Production typically takes place in one or more dedicated and secure Machine Learning workspaces.

8. Monitoring

Machine learning engineers monitor components in staging, testing, and production to collect metrics related to changes in performance of the model, data, and infrastructure. They can use those metrics to take action. Model and data monitoring

can include checking for model performance on new images. Infrastructure monitoring might identify slow endpoint response, inadequate compute capacity, or network problems.

9. Data and model monitoring: events and actions

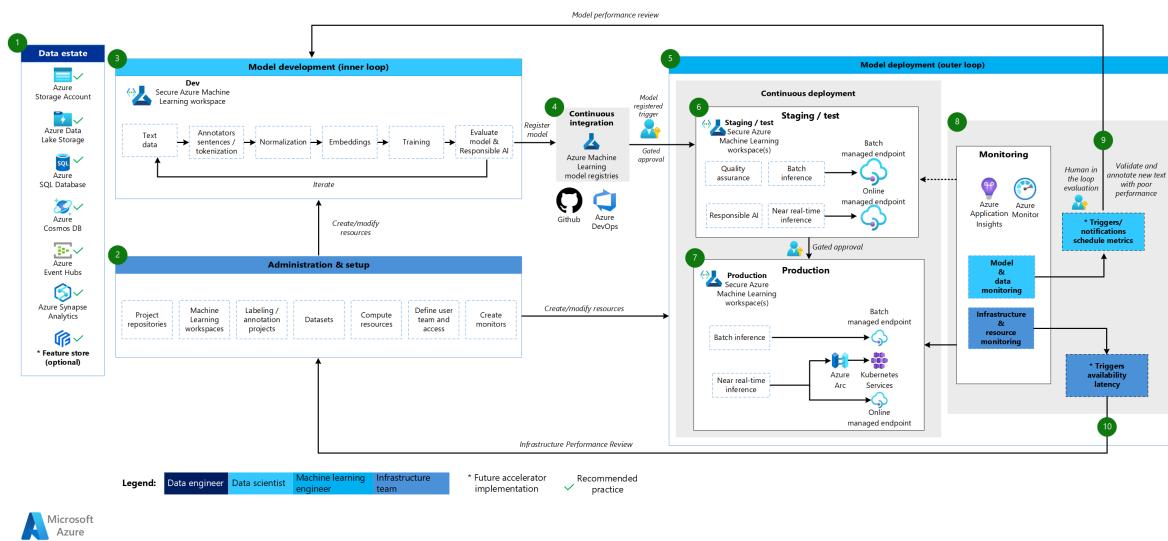
The data and model monitoring and event and action phases of MLOps for natural language processing are the key differences from classical machine learning.

Automated retraining is typically not done in CV scenarios when model performance degradation on new images is detected. In this case, a human-in-the-loop process is necessary to review and annotate new text data for the model that performs poorly. The next action often goes back to the model development loop to update the model with the new images.

10. Infrastructure monitoring: events and actions

Automated triggers and notifications can implement appropriate actions to take based on infrastructure criteria, such as an endpoint response lag or insufficient compute for the deployment. Automatic triggers and notifications might trigger a loopback to the setup and administration phase where the infrastructure team can investigate the problem and potentially reconfigure environment, compute, and network resources.

Machine Learning natural language processing architecture



Download a [Visio file](#) of this architecture.

Workflow for the natural language processing architecture

The Machine Learning natural language processing architecture is based on the classical machine learning architecture, but it has some modifications that are specific to NLP scenarios.

1. Data estate

This component demonstrates the organization data estate and potential data sources and targets for a data science project. Data engineers are the primary owners of this component in the MLOps v2 lifecycle. The Azure data platforms in this diagram aren't exhaustive or prescriptive. A green check mark indicates sources and targets that represent recommended best practices that are based on the customer use case.

2. Administration and setup

This component is the first step in the MLOps v2 accelerator deployment. It consists of all tasks related to the creation and management of resources and roles associated with the project. For natural language processing scenarios, administration and setup of the MLOps v2 environment is largely the same as for classical machine learning, but with an extra step: create image labeling and annotation projects by using the labeling feature of Machine Learning or another tool.

3. Model development (inner loop phase)

The inner loop phase consists of an iterative data science workflow performed within a dedicated and secure Machine Learning workspace. The typical NLP model development loop differs from the classical machine learning scenario in that the typical development steps for this scenario include annotators for sentences and tokenization, normalization, and embeddings for text data.

4. Machine Learning registries

After the data science team develops a model that they can deploy to production, they register the model in the Machine Learning workspace registry. CI pipelines that are triggered automatically by model registration or by gated human-in-the-loop approval promote the model and any other model dependencies to the model deployment phase.

5. Model deployment (outer loop phase)

The model deployment or outer loop phase consists of preproduction staging and testing, production deployment, and monitoring of the model, data, and infrastructure. When the model meets the criteria of the organization and use case, CD pipelines promote the model and related assets through production, monitoring, and potential retraining.

6. Staging and test

The staging and test phase varies according to customer practices. This phase typically includes operations such as retraining and testing the model candidate on production data, test deployments for endpoint performance, data quality checks, unit testing, and responsible AI checks for model and data bias. This phase takes place in one or more dedicated and secure Machine Learning workspaces.

7. Production deployment

After a model passes the staging and test phase, machine learning engineers can use human-in-the-loop gated approval to promote it to production. Model deployment options include a managed batch endpoint for batch scenarios or either a managed online endpoint or Kubernetes deployment that uses Azure Arc for online, near real-time scenarios. Production typically takes place in one or more dedicated and secure Machine Learning workspaces.

8. Monitoring

Machine learning engineers monitor components in staging, testing, and production to collect metrics related to changes in performance of the model, data, and infrastructure. They can use those metrics to take action. Model and data monitoring can include checking for model and data drift, model performance on new text data, and responsible AI problems. Infrastructure monitoring might identify problems, such as slow endpoint response, inadequate compute capacity, and network problems.

9. Data and model monitoring: events and actions

As with the CV architecture, the data and model monitoring and event and action phases of MLOps for natural language processing are the key differences from classical machine learning. Automated retraining isn't typically done in natural language processing scenarios when model performance degradation on new text is detected. In this case, a human-in-the-loop process is necessary to review and annotate new text data for the model that performs poorly. Often the next action is to go back to the model development loop to update the model with the new text data.

10. Infrastructure monitoring: events and actions

Automated triggers and notifications can implement appropriate actions to take based on infrastructure criteria, such as an endpoint response lag or insufficient compute for the deployment. Automatic triggers and notifications might trigger a loopback to the setup and administration phase where the infrastructure team can investigate the problem and potentially reconfigure compute and network resources.

Components

- [Machine Learning](#) is a cloud service that you can use to train, score, deploy, and manage machine learning models at scale.
- [Azure Pipelines](#) is a build-and-test system that's based on Azure DevOps and is used for build and release pipelines. Azure Pipelines splits these pipelines into logical steps called *tasks*.
- [GitHub](#) is a code-hosting platform for version control, collaboration, and CI/CD workflows.
- [Azure Arc](#) is a platform that uses Azure Resource Manager to manage Azure resources and on-premises resources. The resources can include virtual machines, Kubernetes clusters, and databases.
- [Kubernetes](#) is an open-source system that you can use to automate the deployment, scaling, and management of containerized applications.
- [Azure Data Lake Storage](#) is a Hadoop-compatible file system. It has an integrated hierarchical namespace and the massive scale and economy of Blob Storage.
- [Azure Synapse Analytics](#) is a limitless analytics service that brings together data integration, enterprise data warehousing, and big data analytics.
- [Azure Event Hubs](#) is a service that ingests data streams that client applications generate. It then ingests and stores streaming data, which preserves the sequence of events received. Customers can connect to the hub endpoints to retrieve messages for processing. This architecture uses Data Lake Storage integration.

Other considerations

The preceding MLOps v2 architectural pattern has several critical components, including role-based access control (RBAC) that aligns with business stakeholders, efficient package management, and robust monitoring mechanisms. These components collectively contribute to the successful implementation and management of machine learning workflows.

Persona-based RBAC

It's crucial that you manage access to machine learning data and resources. RBAC provides a robust framework to help you manage who can perform specific actions and access specific areas within your solution. Design your identity segmentation strategy to align with the lifecycle of machine learning models in Machine Learning and the personas included in the process. Each persona has a specific set of responsibilities that are reflected in their RBAC roles and group membership.

Example personas

To support appropriate segmentation in a machine learning workload, consider the following common personas that inform the [identity-based RBAC](#) group design.

Data scientist and machine learning engineer

Data scientists and machine learning engineers perform various machine learning and data science activities across the software development lifecycle of a project. Their duties include exploratory data analysis and data preprocessing. Data scientists and machine learning engineers are responsible for training, evaluating, and deploying models. These roles' responsibilities also include break-fix activities for machine learning models, packages, and data. These duties are out of scope for the platform's technical support team.

Type: Person Project specific: Yes

Data analyst

Data analysts provide the necessary input for data science activities, such as running SQL queries for business intelligence. This role's responsibilities include working with data, performing data analysis, and supporting model development and model deployment.

Type: Person Project specific: Yes

Model tester

Model testers conduct tests in testing and staging environments. This role provides functional segregation from the CI/CD processes.

Type: Person Project specific: Yes

Business stakeholders

Business stakeholders are associated with the project, such as a marketing manager.

Type: Person Project specific: Yes

Project lead or data science lead

The data science lead is a project administration role for the Machine Learning workspace. This role also does break-fix activities for the machine learning models and packages.

Type: Person Project specific: Yes

Project or product owner (Business owner)

Business stakeholders are responsible for the Machine Learning workspace according to data ownership.

Type: Person Project specific: Yes

Platform technical support

Platform technical support is the technical support staff responsible for break-fix activities across the platform. This role covers infrastructure or service but not the machine learning models, packages, or data. These components remain under the data scientist or machine learning engineer role and are the project lead's responsibility.

Type: Person Project specific: No

Model end user

Model end users are the end consumers of the machine learning model.

Type: Person or Process Project specific: Yes

CI/CD processes

CI/CD processes release or roll back changes across platform environments.

Type: Process Project specific: No

Machine Learning workspace

Machine Learning workspaces use [managed identities](#) to interact with other parts of Azure. This persona represents the various services that make up a Machine Learning implementation. These services interact with other parts of the platform, such as the development workspace that connects with the development data store.

Type: Process Project specific: No

Monitoring processes

Monitoring processes are compute processes that monitor and alert based on platform activities.

Type: Process Project specific: No

Data governance processes

Data governance processes scan the machine learning project and data stores for data governance.

Type: Process Project specific: No

Microsoft Entra group membership

When you implement RBAC, [Microsoft Entra groups](#) provide a flexible and scalable way to manage access permissions across different personas. You can use Microsoft Entra groups to manage users that need the same access and permissions to resources, such as potentially restricted apps and services. Instead of adding special permissions to individual users, you create a group that applies the special permissions to every member of that group.

In this architectural pattern, you can couple these groups with a [Machine Learning workspace setup](#), such as a project, team, or department. You can associate users with specific groups to define fine-grained access policies. The policies grant or restrict permissions to various Machine Learning workspaces based on job functions, project requirements, or other criteria. For example, you can have a group that grants all data scientists access to a development workspace for a specific use case.

Identity RBAC

Consider how you can use the following built-in Azure RBAC roles to apply RBAC to production and preproduction environments. For the [architecture](#) in this article, the production environments include staging, testing, and production environments. The preproduction environments include development environments. The following RBAC roles are based on the personas described earlier in this article.

Standard roles

- R = [Reader](#)
- C = [Contributor](#)
- O = [Owner](#)

Component specific roles

- ADS = [Machine Learning Data Scientist](#)
- ACO = [Machine Learning Compute Operator](#)
- ACRPush = [Azure Container Registry Push](#)

- DOPA = DevOps Project Administrators
- DOPCA = DevOps Project Collection Administrators
- LAR = Log Analytics Reader
- LAC = Log Analytics Contributor
- MR = Monitoring Reader
- MC = Monitoring Contributor
- KVA = Key Vault Administrator
- KVR = Key Vault Reader

These Azure RBAC role abbreviations correspond with the following tables.

Production environment

[\[+\] Expand table](#)

| Persona | Machine Learning workspace | Azure Key Vault | Container Registry | Azure Storage account | Azure DevOps | Azure Artifacts | Log Analytics workspace | Azure Monitor |
|-------------------------------------|----------------------------|-----------------|--------------------|-----------------------|--------------|-----------------|-------------------------|---------------|
| Data scientist | | R | | | | | LAR | MR |
| Data analyst | | | | | | | | |
| Model tester | | | | | | | | |
| Business stakeholders | | | | | | | | MR |
| Project lead (Data science lead) | R | R, KVR | | | | | LAR | MR |
| Project/product owner | | | | | | | | MR |
| Platform technical support | O | O, KVA | | DOPCA | O | O | O | O |
| Model end user | | | | | | | | |
| CI/CD processes | O | O, KVA | ACRPush | DOPCA | O | O | O | O |

| Persona | Machine Learning workspace | Azure Key Vault | Container Registry | Azure Storage account | Azure DevOps | Azure Artifacts | Log Analytics workspace | Azure Monitor |
|----------------------------|----------------------------|-----------------|--------------------|-----------------------|--------------|-----------------|-------------------------|---------------|
| Machine Learning workspace | | R | C | C | | | | |
| Monitoring processes | R | | | | | | LAR | MR |
| Data governance processes | R | | R | R | R | R | | |

Preproduction environment

[Expand table](#)

| Persona | Machine Learning workspace | Key Vault | Container Registry | Storage account | Azure DevOps | Azure Artifacts | Log Analytics workspace | Azure Monitor |
|-------------------------------------|----------------------------|-----------|--------------------|-----------------|--------------|-----------------|-------------------------|---------------|
| Data scientist | ADS | R, KVA | C | C | C | C | LAC | MC |
| Data analyst | R | | | C | | | LAR | MC |
| Model tester | R | R, KVR | R | R | R | R | LAR | MR |
| Business stakeholders | R | | R | R | R | R | | |
| Project lead (Data science lead) | C | C, KVA | C | C | C | C | LAC | MC |
| Project/product owner | R | | | R | | | | MR |
| Platform technical support | O | O, KVA | O | O | DOPCA | O | O | O |
| Model end user | | | | | | | | |
| CI/CD processes | O | O, KVA | ACRPush | O | DOPCA | O | O | O |

| Persona | Machine Learning workspace | Key Vault | Container Registry | Storage account | Azure DevOps | Azure Artifacts | Log Analytics workspace | Azure Monitor |
|----------------------------|----------------------------|-----------|--------------------|-----------------|--------------|-----------------|-------------------------|---------------|
| Machine Learning workspace | | R, KVR | C | | | | | |
| Monitoring processes | R | R | R | R | R | R | LAC | |
| Data governance processes | R | | R | R | | | | |

ⓘ Note

Every persona retains access for the project's duration except platform technical support, which has temporary or just-in-time [Microsoft Entra Privileged Identity Management \(PIM\)](#) access.

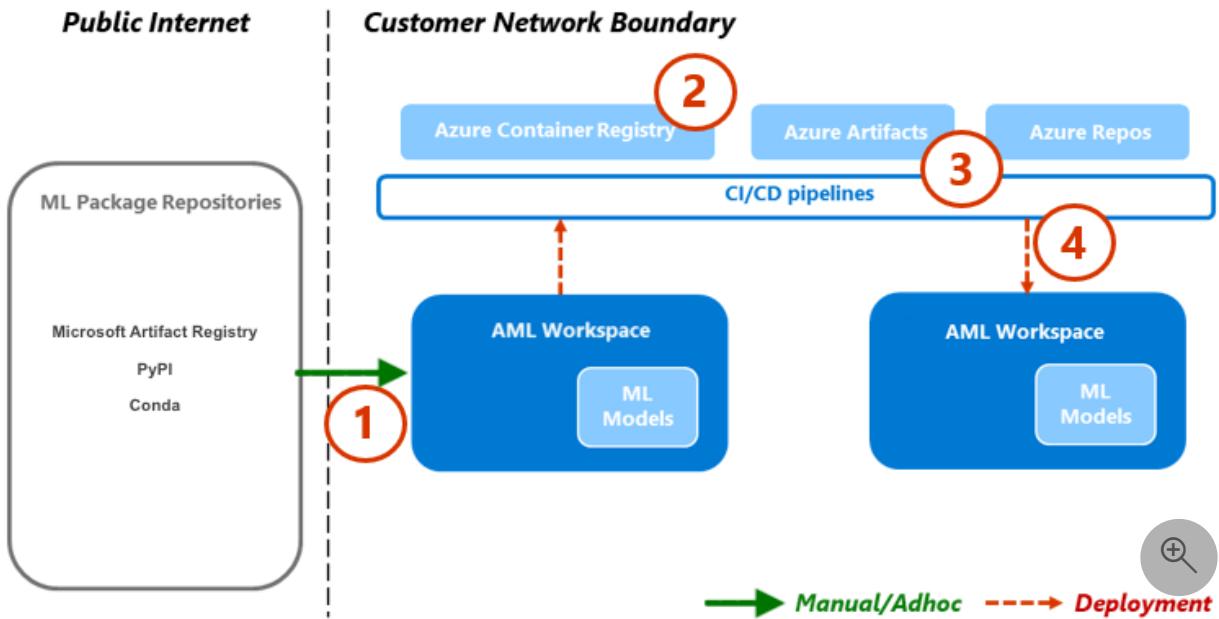
RBAC plays a vital role in securing and streamlining MLOps workflows. RBAC restricts access based on assigned roles and prevents unauthorized users from accessing sensitive data, which mitigates security risks. Sensitive data includes training data or models and critical infrastructure, such as production pipelines. You can use RBAC to ensure compliance with data privacy regulations. RBAC also provides a clear record of access and permissions, which simplifies auditing, makes it easy to identify security gaps, and tracks user activity.

Package management

Dependencies on various packages, libraries, and binaries are common throughout the MLOps lifecycle. These dependencies, often community-developed and rapidly evolving, necessitate subject matter expert knowledge for proper use and understanding. You must ensure that the appropriate people have secure access to diverse assets, such as packages and libraries, but you must also prevent vulnerabilities. Data scientists encounter this problem when they assemble specialized building blocks for machine learning solutions. Traditional software management approaches are costly and inefficient. Other approaches provide more value.

To manage these dependencies, you can use a secure, self-serve, package management process based on the [Quarantine pattern](#). You can design this process to allow data scientists to self-serve from a curated list of packages and ensure that the packages are secure and compliant with organizational standards.

This approach includes safe-listing three industry standard machine learning package repositories: Microsoft Artifact Registry, PyPI, and Conda. Safe-listing enables self-serve from individual Machine Learning workspaces. Then use an automated testing process during the deployment to scan the resulting solution containers. Failures elegantly exit the deployment process and remove the container. The following diagram and process flow demonstrates this process:



Process flow

1. Data scientists that work in a Machine Learning workspace that has a [network configuration](#) can self-serve machine learning packages on-demand from the machine learning package repositories. An exception process is required for everything else by using the [private storage](#) pattern, which is seeded and maintained by using a centralized function.
2. Machine Learning delivers machine learning solutions as docker containers. As these solutions are developed, they're uploaded to Container Registry. [Microsoft Defender for Containers](#) generates vulnerability assessments for the container image.
3. Solution deployment occurs through a CI/CD process. [Microsoft Defender for DevOps](#) is used across the stack to provide security posture management and threat protection.
4. The solution container is deployed only if it passes each of the security processes. If the solution container fails a security process, the deployment fails with error notifications and full audit trails. The solution container is discarded.

The previous process flow provides a secure, self-serve, package management process for data scientists and ensures that the packages are secure and compliant with organizational

standards. To balance innovation and security, you can grant data scientists self-service access to common machine learning packages, libraries, and binaries in preproduction environments. Require exceptions for less common packages. This strategy ensures that data scientists can remain productive during development, which prevents a major bottleneck during delivery.

To streamline your release processes, containerize environments for use in production environments. Containerized environments reduce toil and ensure continued security through vulnerability scanning. This process flow provides a repeatable approach that you can use across use cases to the time of delivery. It reduces the overall cost to build and deploy machine learning solutions within your enterprise.

Monitoring

In MLOps, monitoring is crucial for maintaining the health and performance of machine learning systems and ensuring that models remain effective and aligned with business goals. Monitoring supports governance, security, and cost controls during the inner loop phase. And it provides observability into the performance, model degradation, and usage when deploying solutions during the outer loop phase. Monitoring activities are relevant for personas such as Data Scientists, Business Stakeholders, Project Leads, Project Owners, Platform Technical Support, CI/CD processes, and Monitoring Processes.

Choose your monitoring and verification platform depending on your [Machine Learning workspace setup](#), such as a project, team, or department.

Model performance

Monitor model performance to detect model problems and performance degradation early. Track performance to ensure that models remain accurate, reliable, and aligned with business objectives.

Data drift

[Data drift](#) tracks changes in the distribution of a model's input data by comparing it to the model's training data or recent past production data. These changes are a result of changes in market dynamics, feature transformation changes, or upstream data changes. Such changes can degrade model performance, so it's important to monitor for drift to ensure timely remediation. To perform a comparison, data drift refactoring requires recent production datasets and outputs.

Environment: Production **Azure facilitation:** Machine Learning – [Model monitoring](#)

Prediction drift

Prediction drift tracks changes in the distribution of a model's prediction outputs by comparing it to validation, test-labeled, or recent production data. To perform a comparison, data drift refactoring requires recent production datasets and outputs.

Environment: Production **Azure facilitation:** Machine Learning – [Model monitoring](#)

Resource

Use several model serving endpoint metrics to indicate quality and performance, such as CPU or memory usage. This approach helps you learn from production to help drive future investments or changes.

Environment: All **Azure facilitation:** Monitor - [Online endpoints metrics](#)

Usage metrics

Monitor the usage of endpoints to ensure that you meet organization-specific or workload-specific key performance indicators, track usage patterns, and diagnose and remediate problems that your users experience.

Client requests

Track the number of client requests to the model endpoint to understand the active usage profile of the endpoints, which can affect scaling or cost optimization efforts.

Environment: Production **Azure facilitation:** Monitor - [Online endpoints metrics](#), such as RequestsPerMinute. **Notes:**

- You can align acceptable thresholds to t-shirt sizing or anomalies that are tailored to your workload's needs.
- Retire models that are no longer in use from production.

Throttling delays

[Throttling delays](#) are slowdowns in the request and response of data transfers. Throttling happens at the Resource Manager level and the service level. Track metrics at both levels.

Environment: Production **Azure facilitation:**

- Monitor - [Resource Manager](#), sum of RequestThrottlingDelayMs, ResponseThrottlingDelayMs.
- Machine Learning - To check information about your endpoints' requests, you can enable [online endpoint traffic logs](#). You can use a Log Analytics workspace to process logs.

Notes: Align acceptable thresholds to your workload's service-level objectives (SLOs) or service-level agreements (SLAs) and the solution's nonfunctional requirements (NFRs).

Errors generated

Track response code errors to help measure service reliability and ensure early detection of service problems. For example, a sudden increase in 500 server error responses could indicate a critical problem that needs immediate attention.

Environment: Production **Azure facilitation:** Machine Learning - Enable [online endpoint traffic logs](#) to check information about your request. For example, you can check the count of XRequestId by using ModelStatusCode or ModelStatusReason. You can use a Log Analytics workspace to process logs. **Notes:**

- All HTTP responses codes in the 400 and 500 range are classified as an error.

Cost optimization

Cost management and optimization in a cloud environment are crucial because they help workloads control expenses, allocate resources efficiently, and maximize value from their cloud services.

Workspace compute

When monthly operating expenses reach or exceed a predefined amount, generate alerts to notify relevant stakeholders, such as project leads or project owners, based on the workspace setup boundaries. You can determine your [workspace setup](#) based on project, team, or department-related boundaries.

Environment: All **Azure facilitation:** Microsoft Cost Management - [Budget alerts](#) **Notes:**

- Set budget thresholds based on the initial NFRs and cost estimates.
- Use multiple threshold tiers. Multiple threshold tiers ensure that stakeholders get appropriate warning before the budget is exceeded. These stakeholders might include business leads, project owners, or project Leads depending on the organization or workload.
- Consistent budget alerts could also be a trigger for refactoring to support greater demand.

Workspace staleness

If a Machine Learning workspace shows no signs of active use based on the associated compute usage for the intended use case, a project owner might decommission the

workspace if it's no longer needed for a given project.

Environment: Preproduction Azure facilitation:

- Monitor - [Machine Learning metrics](#)
- Machine Learning - [Workspace metrics](#), such as the count of active cores over a period of time

Notes:

- Active cores should equal zero with aggregation of count.
- Align date thresholds to the project schedule.

Security

Monitor to detect deviations from appropriate security controls and baselines to ensure that Machine Learning workspaces are compliant with your organization's security policies. You can use a combination of predefined and custom-defined policies.

Environment: All Azure facilitation: [Azure Policy for Machine Learning](#)

Endpoint security

To gain visibility into business-critical APIs, implement targeted security monitoring of all Machine Learning endpoints. You can investigate and improve your API security posture, prioritize vulnerability fixes, and quickly detect active real-time threats.

Environment: Production Azure facilitation: [Microsoft Defender for APIs](#) offers broad lifecycle protection, detection, and response coverage for APIs. **Notes:** Defender for APIs provides security for APIs that are published in Azure API Management. You can onboard Defender for APIs in the Microsoft Defender for Cloud portal or within the API Management instance in the Azure portal. You must integrate Machine Learning online endpoints with API Management.

Deployment monitoring

Deployment monitoring ensures that any endpoints you create adhere to your workload or organization policies and are free from vulnerabilities. This process requires that you enforce compliance policies on your Azure resources before and after deployment, provide continued security through vulnerability scanning, and ensure that the service meets SLOs while in operation.

Standards and governance

Monitor to detect deviations from appropriate standards and ensure that your workload adheres to guardrails.

Environment: All Azure facilitation:

- Managed policy assignment and lifecycle through [Azure Pipelines](#) to treat policy as code.
- [PSRule for Azure](#) provides a testing framework for Azure infrastructure as code.
- You can use [Enterprise Azure policy as code](#) in CI/CD-based system deploy policies, policy sets, assignments, policy exemptions, and role assignments.

Notes: For more information, see [Azure guidance for Machine Learning regulatory compliance](#).

Security scanning

Implement automated security scans as part of the automated integration and deployment processes.

Environment: All Azure facilitation: [Defender For DevOps](#) **Notes:** You can use apps in [Azure Marketplace](#) to extend this process for non-Microsoft security testing modules.

Ongoing service

Monitor the ongoing service of an API for performance optimization, security, and resource usage. Ensure timely error detection, efficient troubleshooting, and compliance with standards.

Environment: Production Azure facilitation:

- Monitor - [Machine Learning metrics](#)
- Machine Learning - You can enable [online endpoint traffic logs](#) to check information about your service.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [Scott Donohoo](#) | Senior Cloud Solution Architect
- [Moritz Steller](#) | Senior Cloud Solution Architect

Other contributors:

- [Scott Mckinnon](#) | Cloud Solution Architect
- [Nicholas Moore](#) | Cloud Solution Architect
- [Darren Turchiarelli](#) | Cloud Solution Architect
- [Leo Kozhushnik](#) | Cloud Solution Architect

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- [What is Azure Pipelines?](#)
- [Azure Arc overview](#)
- [What is Machine Learning?](#)
- [Data in Machine Learning](#)
- [Azure MLOps v2 solution accelerator](#)
- [End-to-end machine learning operations \(MLOps\) with Machine Learning](#)
- [Introduction to Azure Data Lake Storage Gen2](#)
- [Azure DevOps documentation](#)
- [GitHub Docs](#)
- [Synapse Analytics documentation](#)
- [Event Hubs documentation](#)
- [How Machine Learning works: resources and assets \(v2\)](#)
- [What are Machine Learning pipelines?](#)

Related resources

- [Choose a Microsoft Azure AI services technology](#)
- [Natural language processing technology](#)
- [Compare the machine learning products and technologies from Microsoft](#)
- [Machine learning operations framework to upscale machine learning lifecycle with Machine Learning](#)
- [What is the Team Data Science Process?](#)

Feedback

Was this page helpful?

 Yes

 No

Generative AI operations for organizations with MLOps investments

This article provides guidance to workload teams that have existing machine learning operations (MLOps) investments and want to extend those investments to include generative AI technology and patterns in their workload. To operationalize generative AI workload features, you need to extend your MLOps investments with generative AI operations (GenAIOps), sometimes known as *LLMOps*. This article outlines technical patterns that are common to both traditional machine learning and generative AI workloads, and patterns unique to generative AI. Understand where you can apply existing investments in operationalization and where you need to extend those investments.

The planning and implementation of MLOps and GenAIOps are part of a core design area in AI workloads on Azure. For more information about why these workloads need specialized operations, see [MLOps and GenAIOps for AI workloads on Azure](#).

Generative AI technical patterns

Generative AI workloads differ from traditional machine learning workloads in several ways:

- **Focus on generative models.** Traditional machine learning workloads focus on training new models for specific tasks. Generative AI workloads consume and sometimes fine-tune generative models that can address a broader range of use cases. Some of these models are multimodal.
- **Focus on extending the models.** The key asset in traditional machine learning is the trained and deployed model. Access to the model is provided to client code in one or more workloads, but the workload typically isn't part of the MLOps process. With generative AI solutions, a key aspect of the solution is the prompt provided to the generative model. The prompt must be composed of instructions and often contains context data from one or more data stores. The system that orchestrates the logic, calls to the various back ends or agents, generates the prompt, and calls to the generative model is part of the generative AI system that you govern with GenAIOps.

Some generative AI solutions use traditional machine learning practices like model training and fine-tuning. However, these solutions introduce new patterns that you should standardize. There are three broad categories of technical patterns for generative AI solutions:

- Fine-tuning
- Prompting
- Retrieval-augmented generation (RAG)

Fine-tuning language models

Many generative AI solutions use existing foundation language models that don't require fine-tuning before use. However, some use cases can benefit from fine-tuning a foundation model, which can be a small language model or large language model.

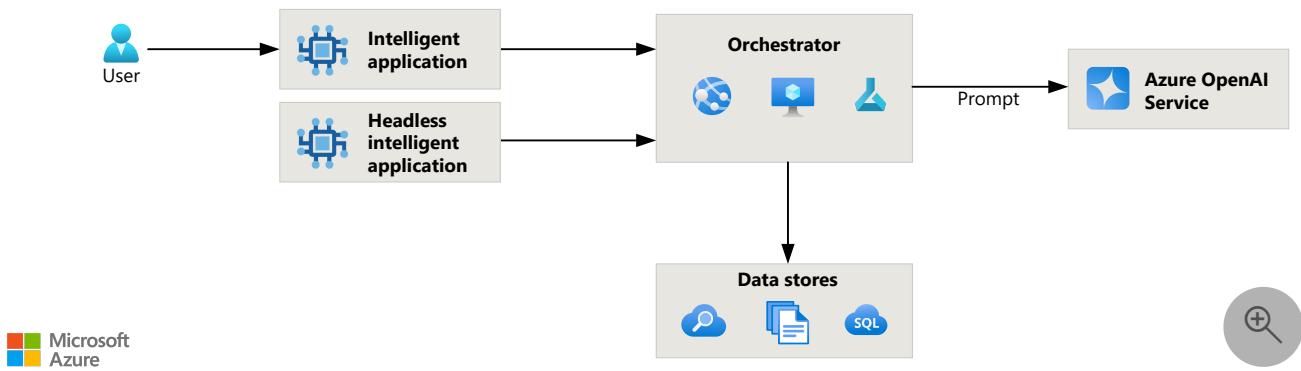
Fine-tuning a foundation model follows logical processes similar to the processes for training traditional machine learning models, such as data preparation, model training, evaluation, and deployment. These processes should use your existing MLOps investments to ensure scalability, reproducibility, and governance.

Prompting

Prompting is the art and science of crafting effective inputs for language models. These inputs typically fall into two categories. System prompts define the model's persona, tone, or behavior. User prompts represent the user's interaction with the language model.

An orchestrator typically manages the workflow that generates these prompts. It can retrieve grounding data from various sources, either directly or via agents, and apply logic to construct the most effective prompt. This orchestrator is often deployed as an API endpoint, which enables client applications to access it as part of an intelligent system.

The following diagram shows an architecture for prompt engineering.



This category of technical patterns can address many use cases:

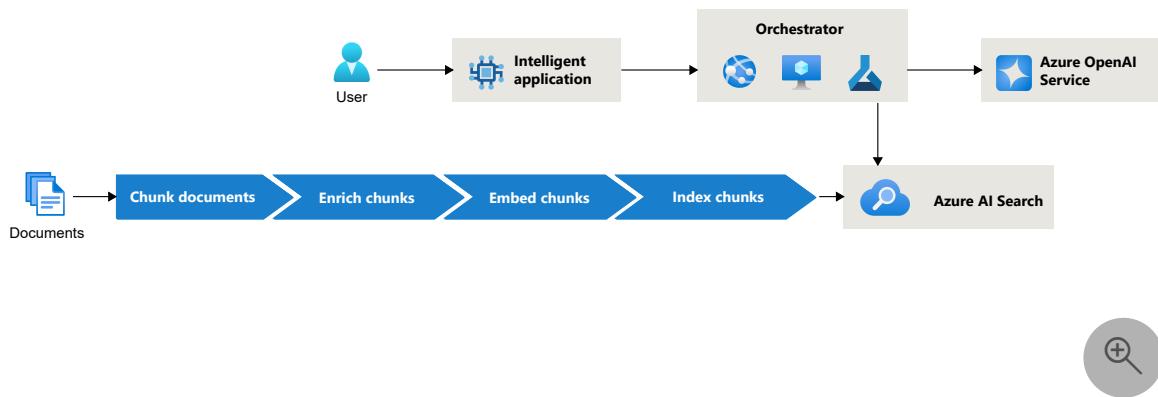
- Classification
- Translation
- Summarization
- RAG

RAG

RAG is an architectural pattern that enhances language models by incorporating domain-specific data into the prompt. This grounding data enables the model to reason over information specific to your company, customers, or domain. In a RAG solution, an orchestration layer queries your data sources and injects the most relevant results into the prompt. The orchestrator then sends this enriched prompt to the language model, typically exposed via an API endpoint for use in intelligent applications.

A typical RAG implementation is to break up your source data into chunks and store them in a vector store along with metadata. Vector stores, such as Azure AI Search, allow you to perform both textual and vector similarity searches to return contextually relevant results. RAG solutions can also [use other data stores](#) to return grounding data.

The following diagram illustrates a RAG architecture that includes data from documents.



Extend MLOps for generative AI technical patterns

Your MLOps process addresses both inner loop and outer loop processes. Generative AI technical patterns also have many of the same activities. In some cases, you apply your existing MLOps investments. In other cases, you need to extend them:

- Inner loop
 - [DataOps](#)
 - [Experimentation](#)
 - [Evaluation](#)
- Outer loop
 - [Deployment](#)
 - [Inferencing and monitoring](#)
 - [Feedback loop](#)

DataOps

Both MLOps and GenAIOps apply the fundamentals of data operations (DataOps) to create extensible and reproducible workflows. These workflows ensure that data is cleaned, transformed, and formatted correctly for experimentation and evaluation. Workflow reproducibility and data versioning are important features of DataOps for all technical patterns. The sources, types, and intent of the data depend on the pattern.

Training and fine-tuning

This technical pattern should fully take advantage of the existing DataOps investments from your MLOps implementation. Reproducibility and data versioning allow you to experiment with different feature engineering data, compare the performance of the different models, and reproduce results.

RAG and prompt engineering

The intent for the data in RAG solutions is to provide grounding data (or context) that's presented to the language model as part of a prompt. RAG solutions often require the processing of large documents or data sets into a collection of right-sized, semantically relevant chunks, and persisting those chunks in a vector store. For more information, see [Design and develop a RAG solution](#). Reproducibility and data versioning for RAG solutions allows you to experiment with different chunking and embedding strategies, compare performance, and roll back to previous versions.

Data pipelines for chunking documents aren't part of DataOps in traditional MLOps, so you have to extend your architecture and operations. The data pipelines can read data from disparate sources that include both structured and unstructured data. They can also write the transformed data to different targets. You must extend your pipelines to include the data stores that you use for grounding data. Typical data stores for these patterns are vector stores like AI Search.

Just like training and fine-tuning, Azure Machine Learning pipelines or other data pipelining tools can be used to orchestrate the stages of chunking.

Search index maintenance

You also must extend your operations to maintain the freshness and validity of the search indexes in your data stores. You might need to periodically rebuild these indexes if you can't incrementally add, remove, or update data in place. Index updates must meet the business requirements for data freshness, the nonfunctional requirements, such as performance and availability, and the compliance requirements, such as *right to be forgotten* requests. You need

to extend your existing MLOps process to account for maintaining and updating search indexes to ensure accuracy, compliance, and optimal performance.

Experimentation

Experimentation, a part of the inner loop, is the iterative process of creating, [evaluating](#), and refining your solution. The following sections describe experimentation for the typical generative AI technical patterns.

Training and fine-tuning

When you fine-tune an existing language model or train a small language model, you can take advantage of your current MLOps investments. For instance, Machine Learning pipelines provide a toolkit for conducting experiments efficiently and effectively. These pipelines enable you to manage the entire fine-tuning process, from data preprocessing to model training and evaluation.

RAG and prompt engineering

Experimentation with prompt engineering and RAG workloads requires you to extend your MLOps investments. For these technical patterns, the workload doesn't end with the model. The workload requires an orchestrator, which is a system that can run logic, call data stores or agents for required information like grounding data, generate prompts, and call language models. The data stores and indexes in the stores are also part of the workload. You need to extend your operations to govern these aspects of the workload.

You can experiment on multiple dimensions for different prompts, including different system instructions, personas, examples, constraints, and advanced techniques like prompt chaining. When you [experiment with RAG solutions](#), you can also experiment with other areas:

- Chunking strategies
- Methods for enriching chunks
- Embedding model selection
- Configuration of the search index
- Types of searches to perform, such as vector, full-text, and hybrid

As described in [DataOps](#), reproducibility and data versioning are key to experimentation. A good experimentation framework enables you to store inputs, such as changes to hyperparameters or prompts, along with outputs to be used when you [evaluate the experiment](#).

Just like in your existing MLOps environment, you can take advantage of frameworks such as Machine Learning pipelines. Machine Learning pipelines have features that support indexing by integrating with vector stores like AI Search. Your GenAIOps environment can take advantage of these pipeline features.

Evaluation and experimentation

Evaluation is key in the iterative experimentation process of building, evaluating, and refining your solution. The evaluation of your changes provides the feedback that you need to make your refinements or validate that the current iteration meets your requirements. The following sections describe evaluation in the experimentation phase for the typical generative AI technical patterns.

Fine-tuning

To evaluate fine-tuned or trained generative AI models, take advantage of your existing MLOps investments. For example, if you use Machine Learning pipelines to orchestrate your machine learning model training, you can use the same evaluation features to fine-tune foundation language models or train new small language models. These features include the [Evaluate Model component](#), which computes industry-standard evaluation metrics for specific model types and compares results across models. If your workload uses Azure AI Foundry, you could instead extend your MLOps process to include its [evaluation capabilities](#) found in the Evaluation SDK.

RAG and prompting

You need to extend your existing MLOps investments to evaluate generative AI solutions. You can use the Evaluations within AI Foundry or our Evaluation SDK.

The experimentation process remains consistent, regardless of the use case for your generative AI solution. These use cases include classification, summarization, translation, and RAG. The important difference is the metrics that you use to evaluate the different use cases. Consider the following metrics based on use case:

- Translation: BLEU
- Summarization: ROUGE, BLEU, BERTScore, METEOR
- Classification: Precision, Recall, Accuracy, Cross-entropy
- RAG: Groundedness, Relevance, Coherence, Fluency

 **Note**

For more information about how to evaluate language models and RAG solutions, see [Large language model end-to-end evaluation](#).

Generative AI solutions generally extend the responsibilities of the machine learning team from training models to prompting and managing grounding data. Because prompting and RAG experimentation and evaluation don't necessarily require data scientists, you might be tempted to use other roles, like software engineers and data engineers, to perform these functions. You might encounter challenges if you omit data scientists from the process of experimenting with prompting and RAG solutions. Other roles often lack the specialized training needed to scientifically evaluate results as effectively as data scientists. For more information, see [Design and develop a RAG solution](#).

Investing in generative AI solutions helps reduce some of the workload on your data science resources. The role of software engineers expands in these solutions. For example, software engineers are great resources for managing the orchestration responsibility in generative AI solutions, and they're adept at setting up the evaluation metrics. It's important to have data scientists review this work. They have the training and experience to understand how to properly evaluate the experiments.

It's also a good idea to request feedback from subject matter experts when you perform evaluations during the initial phase of the project.

Deployment

Some generative AI solutions include deploying custom-trained models or fine-tuning existing models. For generative AI solutions, you need to include the extra tasks of deploying the orchestrators and any data stores. The following sections describe deployment for typical generative AI technical patterns.

Fine-tuning

Use your existing MLOps investments, with some possible adjustments, to deploy generative AI models and fine-tune foundation models. For example, to fine-tune a large language model in Azure OpenAI, ensure that your training and validation datasets are in JSONL format, and upload the data via a REST API. Also create a fine-tuning job. To deploy a trained small language model, take advantage of your existing MLOps investments.

RAG and prompting

For RAG and prompting, consider orchestration logic, modifications to data stores such as indexes and schemas, and adjustments to data pipeline logic. Orchestration logic is typically

encapsulated in a framework like the Microsoft Agent Framework SDK. You can deploy the orchestrator to different compute resources, including resources where you currently deploy custom models. Also, agent orchestrators can be low-code solutions, such as the Azure AI Foundry Agent Service. For more information about how to deploy a chat agent, see [Baseline AI Foundry chat reference architecture](#).

Deployments of changes to database resources, like changes to data models or indexes, are new tasks that need to be handled in GenAIOps. A common practice when working with large language models is to [use a gateway in front of the large language model](#).

Many generative AI architectures that consume platform-hosted language models, like those served from Azure OpenAI, include a [gateway like Azure API Management](#). The gateway use cases include load balancing, authentication, and monitoring. The gateway can play a role in deployment of newly trained or fine-tuned models, which allows you to progressively roll out new models. The use of a gateway, along with model versioning, enables you to minimize risk when you deploy changes and to roll back to previous versions when problems occur.

Deployments of elements that are specific to generative AI, such as the orchestrator, should follow proper operational procedures:

- Rigorous testing, including unit tests
- Integration tests
- A/B tests
- End-to-end tests
- Roll-out strategies, like canary deployments or blue-green deployments

Because the deployment responsibilities for generative AI applications extend beyond model deployment, you might need extra job roles to manage the deployment and monitoring of components like the user interface, the orchestrator, and the data stores. These roles are often aligned to DevOps engineer skill sets.

Inferencing and monitoring

Inferencing is the process of passing input to a trained and deployed model, which then generates a response. You should monitor both traditional machine learning and generative AI solutions from the perspectives of operational monitoring, learning from production, and resource management.

Operational monitoring

Operational monitoring is the process of observing the ongoing operations of the system, including DataOps and model training. This type of monitoring looks for deviations, including

errors, changes to error rates, and changes to processing times.

For model training and fine-tuning, you generally observe the DataOps for processing feature data, model training, and fine-tuning. The monitoring of these inner-loop processes should take advantage of your existing MLOps and DataOps investments.

For prompting in generative AI solutions, you have extra monitoring concerns. You must monitor the data pipelines that process the grounding data or other data that's used to generate prompts. This processing might include data store operations like building or rebuilding indexes.

In a multi-agent system, you need to monitor the availability, performance characteristics, and response quality and consistency of the agents that your orchestrator interfaces with.

As part of operational monitoring, it's important to track metrics such as latency, token usage, and 429 errors to ensure that users aren't encountering significant problems.

Learn from production

A crucial aspect of monitoring during the inferencing stage is learning from production. Monitoring for traditional machine learning models tracks metrics like accuracy, precision, and recall. A key goal is to avoid prediction drift. Solutions that use generative models, such as a GPT model for classification, can take advantage of existing MLOps monitoring investments.

Solutions that use generative models to reason over grounding data use [metrics](#) like groundedness, completeness, usage, and relevancy. The goal is to ensure that the model fully answers the query and bases the response on its context. In this solution, you need to try to prevent problems like data drift. You want to ensure that the grounding data and the prompt that you provide to the model are maximally relevant to the user query.

Solutions that use generative models for nonpredictive tasks, like RAG solutions, often benefit from human feedback from users to evaluate usefulness sentiments. User interfaces can capture feedback like thumbs up or down. You can use this data to periodically evaluate the responses.

A typical pattern for generative AI solutions is to [deploy a gateway in front of the generative models](#). One of the use cases for the gateway is to [monitor the foundation models](#). You can use the gateway to log input prompts and model output.

Another key area to monitor for generative solutions is content safety. The goal is to moderate responses and detect harmful or undesirable content. [Microsoft Azure AI Content Safety Studio](#) is a tool that you can use to moderate content.

Resource management

Generative solutions that use models exposed as a service, like Azure OpenAI, have different resource management concerns than models that you deploy yourself. For models that are exposed as a service, infrastructure management isn't a concern. Instead, the focus is on service throughput, quota, and throttling. Azure OpenAI uses tokens for billing, throttling, and quotas. You should monitor quota usage for cost management and performance efficiency. Azure OpenAI also provides logging capabilities to track token usage.

Tooling

Many MLOps practitioners use a standardized toolkit to organize activities such as automation, tracking, deployment, and experimentation. This approach abstracts common concerns and implementation details, which makes these processes more efficient and manageable. A popular unified platform is [MLflow](#). Before you look for new tools to support GenAIOps patterns, you should review your existing MLOps tooling to evaluate its support for generative AI. For example, MLflow supports a [wide range of features for language models](#).

You can also explore the benefits and trade-offs of introducing new tools into your flow. For example, the [Azure AI Evaluation SDK](#) for Python could be a feasible option because it has native support in the Azure AI Foundry portal.

MLOps and GenAIOps maturity models

You might have used the [MLOps maturity model](#) to evaluate the maturity of your current MLOps and environment. As you extend your MLOps investments for generative AI workloads, you should use the [GenAIOps maturity model](#) to evaluate those operations. You might want to combine the two maturity models, but we recommend that you measure each model independently because MLOps and GenAIOps evolve separately. For example, you might be at level four in the MLOps maturity model but only at level one in the GenAIOps maturity model.

Use the [GenAIOps Maturity Model assessment](#). This assessment helps you understand how your investments in GenAIOps are progressing.

Summary

As you start to extend your MLOps investments to include generative AI, it's important to understand that you don't need to start over. You can use your existing MLOps investments for several of the generative AI technical patterns. Fine-tuning generative models is a great example. Some processes in generative AI solutions, such as prompt engineering and RAG, are

new. Because they're not part of traditional AI workflows, you need to extend your existing operations investments and gain new skills to effectively use them.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

- [Luiz Braz](#) | Senior Technical Specialist
- [Marco Aurelio Cardoso](#) | Senior Software Engineer
- [Paulo Lacerda](#) | Cloud Solution Architect
- [Ritesh Modi](#) | Principal Software Engineer

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Machine Learning](#)
- [Azure OpenAI](#)

Related resources

- [Design and develop a RAG solution](#)
- [Baseline AI Foundry chat reference architecture](#)
- [MLOps](#)

Last updated on 10/17/2025

MLOps maturity model

The machine learning operations (MLOps) maturity model defines principles and practices to help you build and operate production machine learning environments. Use this model to assess your current state and plan incremental progress toward a mature MLOps environment.

Maturity model overview

The MLOps maturity model clarifies the development operations (DevOps) principles and practices required to run a successful MLOps environment. It provides a framework to measure your organization's MLOps capabilities and identify gaps in your current implementation. Use this model to develop your MLOps capability gradually instead of facing the full complexity of mature implementation upfront.

Use the MLOps maturity model as a guide to do the following tasks:

- Estimate the scope of the work for new engagements.
- Establish realistic success criteria.
- Identify deliverables to hand over at the end of the engagement.

Like most maturity models, the MLOps maturity model qualitatively assesses people and culture, processes and structures, and objects and technology. As the maturity level increases, the likelihood that incidents or errors lead to improvements in development and production processes also increases.

The MLOps maturity model encompasses five levels of technical capability.

[] Expand table

| Level | Description | Highlights | Technology |
|-------|-------------|---|---|
| 0 | No MLOps | <ul style="list-style-type: none">• Full machine learning model life cycle is difficult to manage.• Teams are disparate and releases are challenging.• Most systems are nontransparent, with little feedback during and after deployment. | <ul style="list-style-type: none">• Builds and deployments are manual.• Model and application testing is manual.• Model performance tracking isn't centralized.• Model training is manual. |

| Level | Description | Highlights | Technology |
|-------|--|--|--|
| | | | <ul style="list-style-type: none"> Teams use only basic Azure Machine Learning workspace features. |
| 1 | DevOps but no MLOps | <ul style="list-style-type: none"> Releases are less challenging than Level 0, but rely on data teams for every new model. Feedback about model performance in production is still limited. Results are difficult to trace and reproduce. | <ul style="list-style-type: none"> Builds are automated. Application code has automated tests. Code is version controlled. |
| 2 | Automated training | <ul style="list-style-type: none"> Training environment is fully managed and traceable. Model is easy to reproduce. Releases are manual but easy to implement. | <ul style="list-style-type: none"> Model training is automated. Model training performance tracking is centralized. Model management is in place. Machine Learning scheduled or event-driven jobs handle recurring training. Managed feature store is adopted. Azure Event Grid life cycle events are emitted for pipeline orchestration. Environments are managed by using Machine Learning environment definitions. |
| 3 | Automated model deployment | <ul style="list-style-type: none"> Releases are easy to implement and automatic. Full traceability exists from deployment back to original data. | <ul style="list-style-type: none"> A/B testing of model performance is integrated for deployment. All code has automated tests. |

| Level | Description | Highlights | Technology |
|-------|---------------------------------|---|--|
| | | <ul style="list-style-type: none"> Entire environment is managed, including training, testing, and production. | <ul style="list-style-type: none"> Model training performance tracking is centralized. Artifacts are promoted across workspaces by using Machine Learning registries. |
| 4 | Full MLOps automated operations | <ul style="list-style-type: none"> Full system is automated and easily monitored. Production systems provide information about how to improve, and sometimes automatically improve with new models. System is approaching zero downtime. | <ul style="list-style-type: none"> Model training and testing are automated. Deployed model emits verbose, centralized metrics. Drift or regression signals trigger automatic retraining by using Event Grid. Feature materialization health and freshness are monitored. Model promotion is policy-based and automated by using Machine Learning registries. |

The following tables describe detailed characteristics for each level of maturity.

Level 0: No MLOps

 Expand table

| People | Model creation | Model release | Application integration |
|---|--|--|---|
| <ul style="list-style-type: none"> Data scientists work in isolation without regular communication with the larger team. Data engineers (if they exist) work in isolation without | <ul style="list-style-type: none"> Data is gathered manually. Compute is likely not managed. Experiments aren't tracked | <ul style="list-style-type: none"> Release process is manual. Scoring script is created manually after experiments and isn't version | <ul style="list-style-type: none"> Implementation depends heavily on data scientist expertise. Application releases are manual. |

| People | Model creation | Model release | Application integration |
|--|---|--|-------------------------|
| <p>regular communication with the larger team.</p> <ul style="list-style-type: none"> Software engineers work in isolation and receive models remotely from other team members. | <p>consistently.</p> <ul style="list-style-type: none"> End result is typically a single model file that includes inputs and outputs, handed off manually. | <p>controlled.</p> <ul style="list-style-type: none"> A single data scientist or data engineer handles release. | |

Level 1: DevOps but no MLOps

[Expand table](#)

| People | Model creation | Model release | Application integration |
|--|---|--|--|
| <ul style="list-style-type: none"> Data scientists work in isolation without regular communication with the larger team. Data engineers (if they exist) work in isolation without regular communication with the larger team. Software engineers work in isolation and receive models remotely from other team members. | <ul style="list-style-type: none"> Data pipeline automatically gathers data. Compute might or might not be managed. Experiments aren't tracked consistently. End result is typically a single model file that includes inputs and outputs, handed off manually. | <ul style="list-style-type: none"> Release process is manual. Scoring script is created manually after experiments but is likely version controlled. Model is handed off to software engineers. | <ul style="list-style-type: none"> Basic integration tests exist for the model. Implementation depends heavily on data scientist expertise. Application releases are automated. Application code has unit tests. |

Level 2: Automated training

[Expand table](#)

| People | Model creation | Model release | Application integration |
|--|--|--|---|
| <ul style="list-style-type: none"> Data scientists work directly with data engineers to convert experimentation code into repeatable scripts and jobs. Data engineers work with data scientists on model development. Software engineers work in isolation and receive models remotely from other team members. | <ul style="list-style-type: none"> Data pipeline automatically gathers data. Compute is managed. Experiment results are tracked. Training code and models are both version controlled. | <ul style="list-style-type: none"> Release process is manual. Scoring script is version controlled and has tests. Software engineering team manages releases. | <ul style="list-style-type: none"> Basic integration tests exist for the model. Implementation depends heavily on data scientist expertise. Application code has unit tests. |

Level 3: Automated model deployment

 Expand table

| People | Model creation | Model release | Application integration |
|--|--|---|--|
| <ul style="list-style-type: none"> Data scientists work directly with data engineers to convert experimentation code into repeatable scripts and jobs. Data engineers work with data scientists and software engineers to manage inputs and outputs. Software engineers work with data engineers to automate model integration into application code. | <ul style="list-style-type: none"> Data pipeline automatically gathers data. Compute is managed. Experiment results are tracked. Training code and models are both version controlled. | <ul style="list-style-type: none"> Release process is automatic. Scoring script is version controlled and has tests. Continuous integration and continuous delivery (CI/CD) pipeline manages releases. | <ul style="list-style-type: none"> Each model release includes unit and integration tests. Implementation is less dependent on data scientist expertise. Application code has unit and integration tests. |

Level 4: Full MLOps automated operations

 Expand table

| People | Model creation | Model release | Application integration |
|---|--|--|--|
| <ul style="list-style-type: none">• Data scientists work directly with data engineers to convert experimentation code into repeatable scripts and jobs. They also work with software engineers to identify data markers.• Data engineers work with data scientists and software engineers to manage inputs and outputs.• Software engineers work with data engineers to automate model integration and implement post-deployment metrics gathering. | <ul style="list-style-type: none">• Data pipeline automatically gathers data.• Production metrics automatically trigger retraining.• Compute is managed.• Experiment results are tracked.• Training code and models are both version controlled. | <ul style="list-style-type: none">• Release process is automatic.• Scoring script is version controlled and has tests.• CI/CD pipeline manages releases. | <ul style="list-style-type: none">• Each model release includes unit and integration tests.• Implementation is less dependent on data scientist expertise.• Application code has unit and integration tests. |

MLOps and GenAIOps

This article focuses on predictive, tabular, and classical machine learning life cycle capabilities. Generative AI operations (GenAIOps) introduce extra capabilities that complement the MLOps maturity levels rather than replace them. GenAIOps include prompt life cycle, retrieval augmentation, output safety, and token cost governance. For more information, see [GenAIOps for organizations that have MLOps investments](#). Don't confuse prompt iteration mechanics with the reproducible training-deployment loop described in this article.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

- [Delyn Choong](#) | Senior Cloud Solutions Architect – Data & AI

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- MLOps and GenAIOps for AI workloads
- Learning path: Introduction to MLOps
- MLOps model management, deployment, and monitoring by using Machine Learning
- Machine learning registries for MLOps

Related resources

- [Orchestrate MLOps by using Azure Databricks](#)
- [MLOps overview](#)

Last updated on 11/20/2025

Access Azure OpenAI and other language models through a gateway

Azure AI services

Azure OpenAI Service

Azure API Management

This article describes the key challenges across the five pillars of the [Azure Well-Architected Framework](#) that you encounter if your workload design includes direct access from your consumers to the Azure OpenAI Service data plane APIs. Learn how introducing a gateway into your architecture can help resolve these direct access challenges, while introducing new challenges. This article describes the architectural pattern but not how to implement the gateway.

[Azure OpenAI](#) exposes HTTP APIs that let your applications perform embeddings or completions by using OpenAI's language models. Intelligent applications call these HTTP APIs directly from clients or orchestrators. Examples of clients include chat UI code and custom data processing pipelines. Examples of orchestrators include Microsoft Agent Framework, Semantic Kernel, LangChain, and Microsoft Foundry Agent Service. When your workload connects to one or more Azure OpenAI instances, you must decide whether these consumers connect directly or through a reverse proxy API gateway.

Because a gateway can be used to solve specific scenarios that might not be present in every workload, see be sure to see [Specific scenario guidance](#), which looks at that specific use case of a gateway in more depth.

Key challenges

Without an API gateway or the ability to add logic into the Azure OpenAI HTTP APIs, the client has to handle the API client logic, which includes retry mechanisms or circuit breakers. This situation can be challenging in scenarios in which you don't directly control the client code, or when the code is restricted to specific SDK usage. Multiple clients or multiple Azure OpenAI instances and deployments present further challenges, such as coordination of safe deployments and observability.

This section provides examples of specific key architectural challenges that you might face if your architecture only supports direct access to Azure OpenAI from consumers. The challenges are organized by using the [Azure Well-Architected Framework pillars](#).

Reliability challenges

The reliability of the workload depends on several factors, including its capacity for self-preservation and self-recovery, which are often implemented through replication and failover

mechanisms. Without a gateway, all reliability concerns must be addressed exclusively by using client logic and Azure OpenAI Service features. Workload reliability is compromised when there isn't enough reliability control available in either of those two surfaces.

- **Load balancing or Redundancy:** Failing over between multiple Azure OpenAI instances based on service availability is a client responsibility that you need to control through configuration and custom logic.

Whether you use [Global](#), standard or provisioned, or [data zone](#), standard or provisioned, it doesn't affect the Azure OpenAI service availability from a regional endpoint availability perspective. You still have a responsibility to implement failover logic yourself.

- **Scale out to handle spikes:** Failing over to Azure OpenAI instances with capacity when throttled is another client responsibility that you need to control through configuration and custom logic. Updating multiple client configurations for new Azure OpenAI instances presents greater risk and has timeliness concerns. The same is true for updating client code to implement changes in logic, such as directing low priority requests to a queue during high demand periods.
- **Throttling:** Azure OpenAI APIs throttle requests by returning an HTTP 429 error response code to requests that exceed the Token-Per-Minute (TPM) or Requests-Per-Minute (RPM) in the standard model. Azure OpenAI APIs also throttle requests that exceed provisioned capacity for the pre-provisioned billing model. Handling appropriate back-off and retry logic is left exclusively to client implementations.

Most workloads should solve this specific issue by using [global](#) and [data zone](#) deployments of Azure OpenAI. Those deployments to use model capacity from data centers with the enough capacity for each request. Using global and data zone deployments will significantly decrease service throttling without added complexity of custom gateways. The global and data zone deployments are themselves a gateway implementation.

Security challenges

Security controls must help protect workload confidentiality, integrity, and availability. Without a gateway, all security concerns must be addressed exclusively in client logic and Azure OpenAI Service features. Workload requirements might demand more than what's available for client segmentation, client control, or service security features for direct communication.

- **Identity management - authentication scope:** The data plane APIs exposed by Azure OpenAI can be secured in one of two ways: API key or Azure role-based access control (RBAC). In both cases, authentication happens at the Azure OpenAI instance level, not the

individual deployment level, which introduces complexity for providing least privileged access and identity segmentation for specific deployment models.

- **Identity management - identity providers:** Clients that can't use identities located in the Microsoft Entra tenant that's backing the Azure OpenAI instance must share a single full-access API key. API keys have security usefulness limitations and are problematic when multiple clients are involved and all share the same identity.
- **Network security:** Depending on client location relative to your Azure OpenAI instances, public internet access to language models might be necessary.
- **Data sovereignty:** Data sovereignty in the context of Azure OpenAI refers to the legal and regulatory requirements related to the storage and processing of data within the geographic boundaries of a specific country or region. Your workload needs to ensure regional affinity so that clients can comply with data residency and sovereignty laws. This process involves multiple Azure OpenAI deployments.

You should be aware that when you are using [global](#) or [data zone](#) deployments of Azure OpenAI, data at rest remains in the designated Azure geography, but data may be transmitted and processed for inferencing in any Azure OpenAI location.

Cost optimization challenges

Workloads benefit when architectures minimize waste and maximize utility. Strong cost modeling and monitoring are an important requirement for any workload. Without a gateway, utilization of provisioned or per-client cost tracking can be authoritatively achieved exclusively from aggregating Azure OpenAI instance usage telemetry.

- **Cost tracking:** Being able to provide a financial perspective on Azure OpenAI usage is limited to data aggregated from Azure OpenAI instance usage telemetry. When required to do chargeback or showback, you need to attribute that usage telemetry with various clients across different departments or even customers for multitenant scenarios.
- **Provisioned throughput utilization:** Your workload wants to avoid waste by fully utilizing the provisioned throughput that you paid for. This means that clients must be trusted and coordinated to use provisioned model deployments before spilling over into any standard model deployments.

Operational excellence challenges

Without a gateway, observability, change control, and development processes are limited to what is provided by direct client-to-server communication.

- **Quota control:** Clients receive 429 response codes directly from Azure OpenAI when the HTTP APIs are throttled. Workload operators are responsible for ensuring that enough quota is available for legitimate usage and that misbehaving clients don't consume in excess. When your workload consists of multiple model deployments or multiple data zones, understanding quota usage and quota availability can be difficult to visualize.
- **Monitoring and observability:** Azure OpenAI default metrics are available through Azure Monitor. However, there's latency with the availability of the data and it doesn't provide real-time monitoring.
- **Safe deployment practices:** Your GenAIOps process requires coordination between clients and the models that are deployed in Azure OpenAI. For advanced deployment approaches, such as blue-green or canary, logic needs to be handled on the client side.

Performance efficiency challenges

Without a gateway, your workload puts responsibility on clients to be individually well-behaved and to behave fairly with other clients against limited capacity.

- **Performance optimization - priority traffic:** Prioritizing client requests so that high priority clients have preferential access over low priority clients would require extensive, and likely unreasonable, client-to-client coordination. Some workloads might benefit from having low priority requests queued to run when model utilization is low.
- **Performance optimization - client compliance:** To share capacity, clients need to be well-behaved. An example of this is when clients ensure that `max_tokens` and `best_of` are set to approved values. Without a gateway, you must trust clients to act in the best interest of preserving capacity of your Azure OpenAI instance.
- **Minimize latency:** While network latency is usually a small component of the overall prompt and completion request flow, ensuring that clients are routed to a network endpoint and model close to them might be beneficial. Without a gateway, clients would need to self-select which model deployment endpoints to use and what credentials are necessary for that specific Azure OpenAI data plane API.

Solution

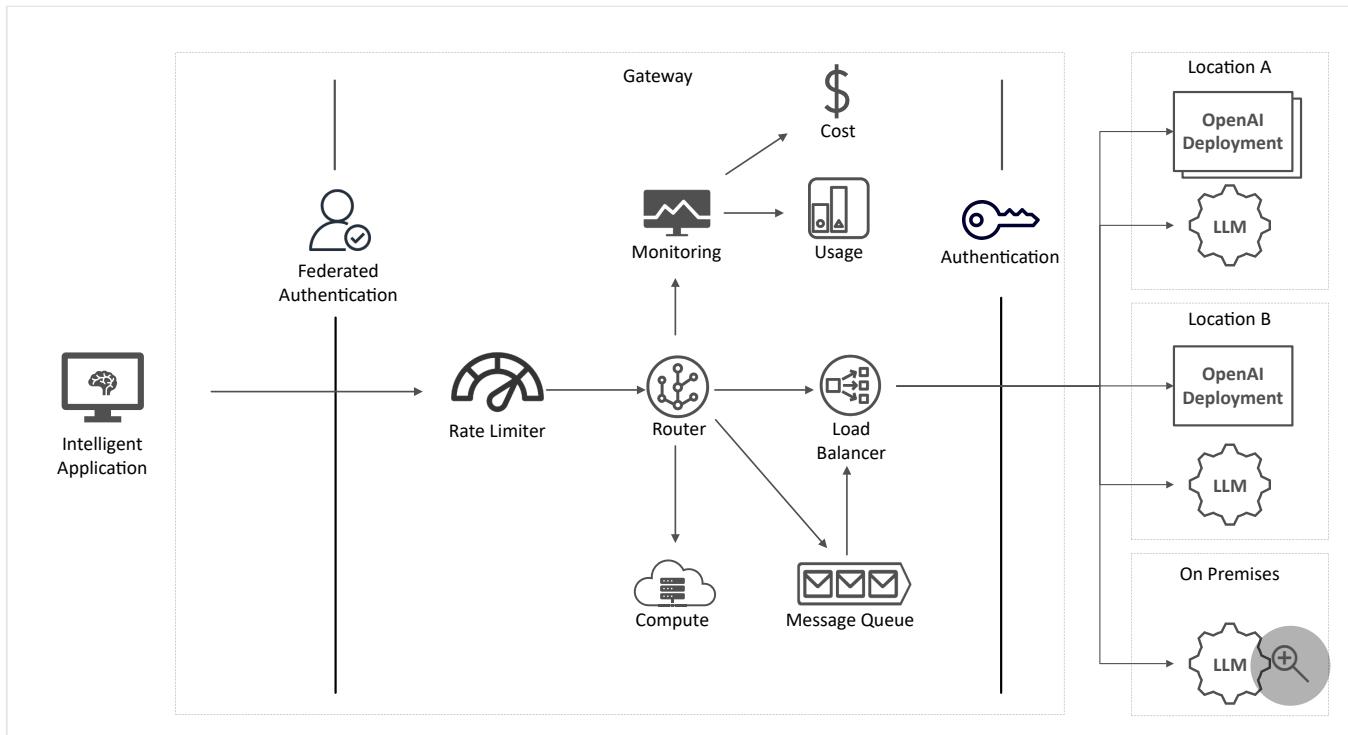


Figure 1: Conceptual architecture of accessing Azure OpenAI through a gateway

To address the many challenges listed in [Key challenges](#), you can inject a reverse proxy gateway to decouple the intelligent application from Azure OpenAI. This [gateway offloading](#) lets you shift responsibility, complexity, and observability away from clients and gives you an opportunity to augment Azure OpenAI by providing other capabilities that aren't built in. Some examples are:

- Potential to implement [federated authentication](#).
- Ability to control pressure on models through [rate limiting](#).
- Cross-cutting and cross-model monitoring.
- Ability to introduce [gateway aggregation](#) and advanced [routing](#) to multiple services, such as routing low priority messages to a queue for [queue-based load leveling](#) or to compute to perform tasks.
- Load balancing that uses [health endpoint monitoring](#) to route only to healthy endpoints by [circuit breaking](#) on unavailable or overloaded model deployments.

Some specific scenarios have more guidance available that directly addresses an API gateway and Azure OpenAI instances. Those scenarios are listed in the [Next steps](#) section.

Considerations

The decision to add a gateway and what technology to use is made as part of the [Application design](#) described in the Azure Well-Architected Framework's [AI workloads on Azure](#) guidance.

As an architect, you'll need to make the decision to include or exclude this component.

When you introduce a new component into your architecture, you need to evaluate the newly introduced tradeoffs. When you inject an API gateway between your clients and the Azure OpenAI data plane to address any of [key challenges](#), you introduce new considerations into your architecture. Carefully evaluate whether the workload impact across these architectural considerations justifies the added value or utility of the gateway.

Reliability

Reliability ensures that your application meets the commitments you make to your customers. For more information, see [Design review checklist for Reliability](#).

- The gateway solution can introduce a single point of failure. This failure could have its origin in service availability of the gateway platform, interruptions due to code or configuration deployments, or even misconfigured critical API endpoints in your gateway. Ensure that you design your implementation to meet your workload's availability requirements. Consider resiliency and fault tolerance capabilities in the implementation by including the gateway in the [failure mode analysis](#) of the workload.
- Your solution might require global routing capabilities if your architecture requires Azure OpenAI instances in multiple regions to increase the availability of your Azure OpenAI endpoints, such as the ability to continue to serve requests in the event of a regional outage. This situation can further complicate the topology through management of extra fully qualified domain names, TLS certificates, and more global routing components.

Important

Don't implement a gateway if doing so would jeopardize your workload's ability to meet agreed upon service-level objectives (SLOs).

Security

When considering how an API gateway benefits your architecture, use the [Design review checklist for Security](#) to evaluate your design. You need to address the following security considerations:

- The surface area of the workload is increased with the addition of the gateway. That surface area brings extra identity and access management (IAM) considerations of the Azure resources, increased hardening efforts, and more.

- The gateway can act as a network boundary transition between client network space and private Azure OpenAI network space. Even though the gateway makes a previously internet-facing Azure OpenAI endpoint private through the use of Azure Private Link, it now becomes the new point of entry and must be adequately secured.
- A gateway is in a unique position to see raw request data and formulated responses from the language model, which could include confidential data from either source. Data compliance and regulatory scope is now extended to this other component.
- A gateway can extend the scope of client authorization and authentication beyond Microsoft Entra ID and API key authentication, and potentially across multiple identity providers (IdP).
- Data sovereignty must be factored in your implementation in multi-region implementations. Ensure that your gateway compute and routing logic adheres to sovereignty requirements placed on your workload.

 **Important**

Don't implement a gateway if doing so would leave your workload unable to protect the confidentiality, integrity, or availability of itself or its users' data.

Cost Optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Design review checklist for Cost Optimization](#).

All implemented API gateways have runtime costs that need to be budgeted and accounted for. Those costs usually increase with added features to address the reliability, security, and performance of the gateway itself along with operational costs introduced with added APIOps management. These added costs need be measured against the new value delivered from the system with the gateway. You want to reach a point where the new capabilities introduced by using a gateway outweigh the cost to implement and maintain the gateway. Depending on your workload's relationship to its users, you might be able to chargeback usage.

To help manage costs when developing and testing a gateway, consider using a simulated endpoint for Azure OpenAI. For example, use the solution in the [Azure OpenAI API simulator](#) GitHub repository.

Operational Excellence

When considering how an API gateway benefits your architecture, use the [Design review checklist for Operational Excellence](#) to evaluate your design. You need to address the following operational excellence considerations:

- The gateway itself needs to be monitored by your workload's monitoring solution and potentially by clients. This means that gateway compute and operations need to be included in the workload's [health modeling](#).
- Your safe deployment practices now need to address the deployment of the API gateway infrastructure and the code or configuration of the gateway routing. Your infrastructure automation and infrastructure as code (IaC) solution needs to consider how to treat your gateway as a long-lived resource in the workload.
- You need to build or extend your APIOps approach to cover the APIs exposed in the gateway.
- You duplicate capabilities that are available through solutions such as the Azure AI Service resource or Azure OpenAI data zone load distribution functionality.

Performance Efficiency

When considering how an API gateway benefits your architecture, use the [Design review checklist for Performance Efficiency](#) to evaluate your design. You need to address the following performance efficiency considerations:

- The gateway service can introduce a throughput bottleneck. Ensure the gateway has adequate performance to handle full concurrent load and can easily scale in line with your growth expectations. Ensure elasticity in the solution so that the gateway can reduce supply, or scale down, when demand is low, such as with business day usage.
- The gateway service has processing it must perform per request, and introduces added latency per API invocation. You should optimize your routing logic to keep requests performing well.
- In most cases, the gateway should be geographically near both the users and the Azure OpenAI instances to reduce latency. While network latency is usually a small percentage of time in overall API calls to language models, it might be a competitive factor for your workload.
- Evaluate the impact of the gateway on Azure OpenAI features, such as streaming responses or instance pinning for stateful interactions, such as the Assistants API.



Important

Don't implement a gateway if doing so makes achieving negotiated performance targets impossible or too compromising on other tradeoffs.

Implementation options

Azure doesn't offer a turn-key solution designed specifically to proxy Azure OpenAI's HTTP API or other custom language model inferencing APIs to cover all of these scenarios. But there are still several options for your workload team to implement, such as a gateway in Azure.

Use Azure API Management

[Azure API Management](#) is a platform-managed service designed to offload cross-cutting concerns for HTTP-based APIs. It's configuration driven and supports customization through its inbound and outbound request processing policy system. It supports highly available, zone-redundant, and even multi-region replicas by using a single control plane.

Most of the gateway routing and request handling logic must be implemented in the policy system of API Management. You can combine [built-in policies](#) specific to Azure OpenAI, such as [Limit Azure OpenAI API token usage](#) or [Emit metrics for consumption of Azure OpenAI tokens](#), and your own custom policies. The [GenAI gateway toolkit](#) GitHub repository contains multiple custom API Management policies, along with a load-testing setup for testing the behavior of the policies.

Use the [Well-Architected Framework service guide for API Management](#) when designing a solution that involves Azure API Management. If your workload exists as part of an application landing zone, review the guidance available in the Cloud Adoption Framework for Azure on implementing an [Azure API Management landing zone](#).

Using Azure API Management for your gateway implementation is generally the preferred approach to building and operating an Azure OpenAI gateway. It's preferred because the service is a platform as a service (PaaS) offering with rich built-in capabilities, high availability, and networking options. It also has robust APIOps approaches to managing your completion APIs.

Use custom code

The custom code approach requires a software development team to create a custom coded solution and to deploy that solution to an Azure application platform of their choice. Building a self-managed solution to handle the gateway logic can be a good fit for workload teams proficient at managing network and routing code.

The workload can usually use compute that they're familiar with, such as hosting the gateway code on Azure App Service, Azure Container Apps, or Azure Kubernetes Service.

Custom code deployments can also be fronted with API Management when API Management is used exclusively for its core HTTP API gateway capabilities between your clients and your custom code. This way your custom code interfaces exclusively with your Azure OpenAI HTTP APIs based on the necessary business logic.

The use of non-Microsoft gateway technology, which is a product or service that isn't natively provided by Azure, can be considered as part of this approach.

Example architecture

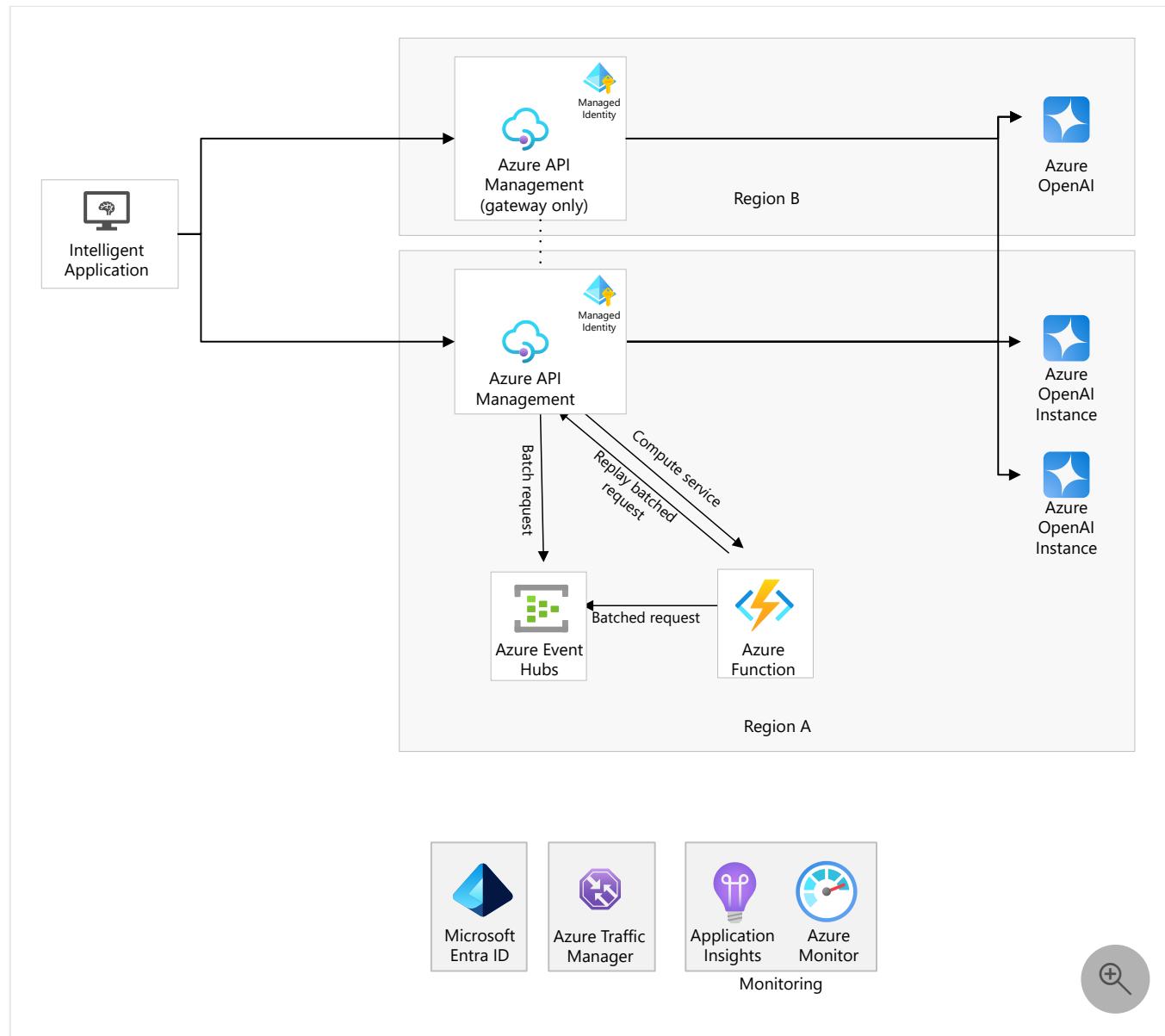


Figure 2: Example architecture of accessing Azure OpenAI through an Azure API Management-based gateway

Next steps

Learn about a specific scenario where deploying a gateway between an intelligent application and Azure OpenAI deployments is used to address workload requirements:

- Load balancing or failover between multiple backend instances
- Custom authentication and authorization for client applications
- Implement logging and monitoring for Azure OpenAI models

Related resources

- [API gateway in Azure API Management](#)
- [API Management landing zone ↗ GitHub repository covering generative AI scenarios](#)
- [API Management gateway toolkit ↗](#)
- [OpenAI API Simulator ↗](#)

Use a gateway in front of multiple Azure OpenAI deployments or instances

Azure AI services

Azure OpenAI Service

Azure API Management

Workload architectures that involve Azure OpenAI Service can be as simple as one or more client applications directly consuming a single Azure OpenAI model deployment directly, but not all workloads can be designed with such simplicity. More complex scenarios include topologies with multiple clients, multiple Azure OpenAI deployments, or multiple Azure OpenAI instances. In those situations, introducing a gateway in front of Azure OpenAI can be beneficial to the workload's design as a programmable routing mechanism.

Multiple Azure OpenAI instances or model deployments solve specific requirements in a workload architecture. They can be classified in four key topologies.

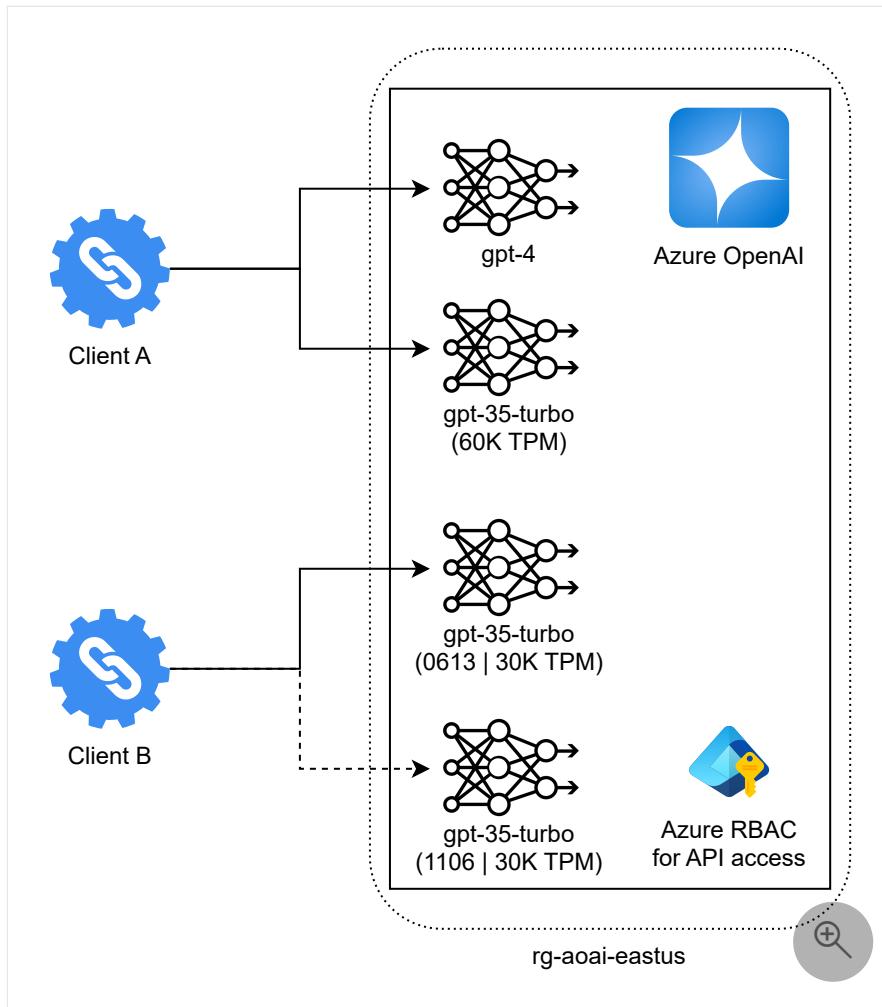
- [Multiple model deployments in a single Azure OpenAI instance](#)
- [Multiple Azure OpenAI instances in a single region and single subscription](#)
- [Multiple Azure OpenAI instances in a single region across multiple subscriptions](#)
- [Multiple Azure OpenAI instances across multiple regions](#)

These topologies on their own don't necessitate the use of a gateway. The choice of a gateway depends on whether the workload benefits from its inclusion in the architecture. This article describes the challenges that each of the four topologies address, and the benefits and costs of including a gateway in each topology.

💡 Tip

Unless otherwise stated, the following guidance is suitable for both Azure API Management-based gateways or custom code gateways. The architecture diagrams represent the gateway component generically in most situations to illustrate this.

Multiple model deployments in a single Azure OpenAI instance



Topology details for multiple model deployments

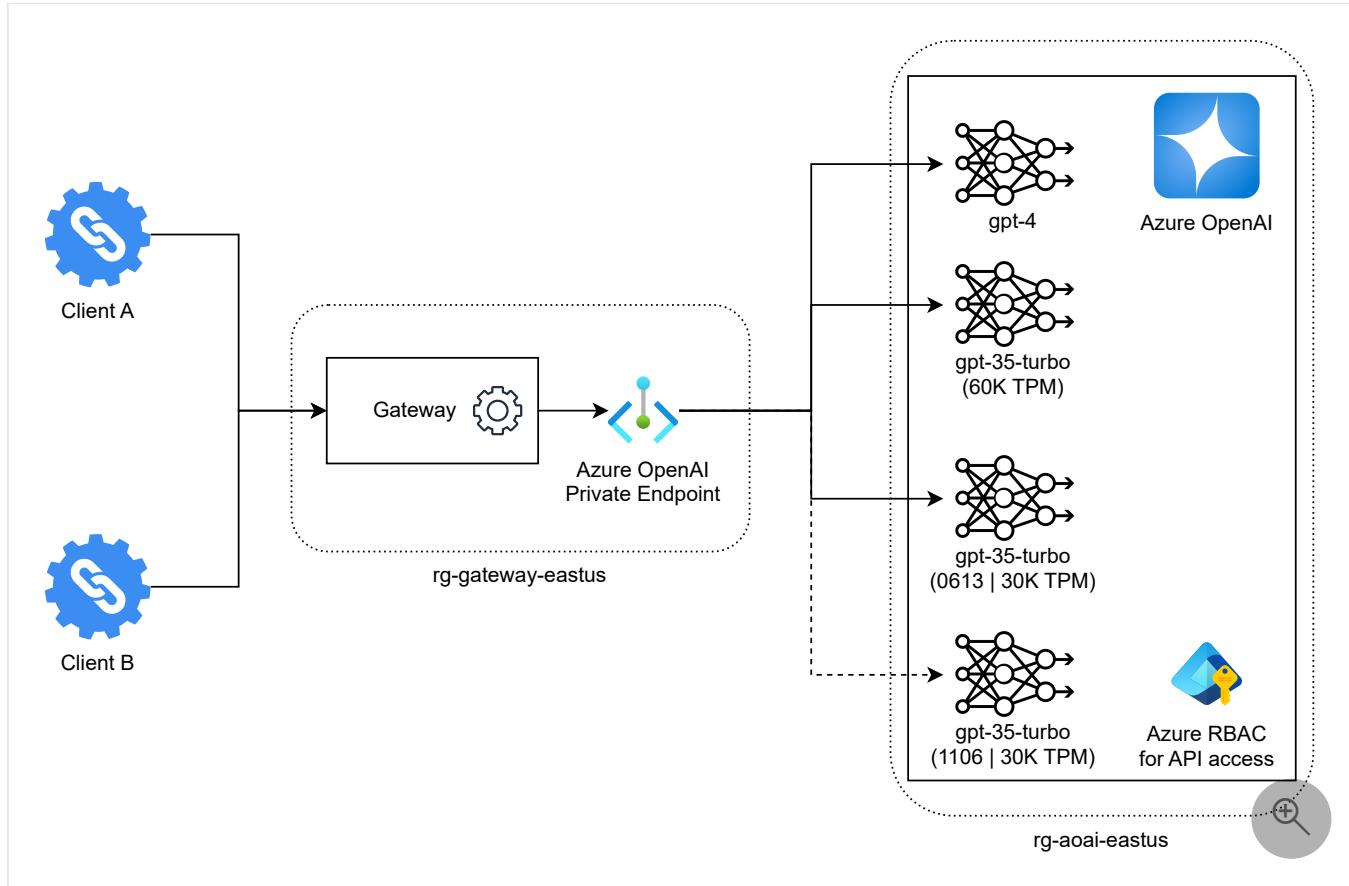
- Azure OpenAI model deployments: multiple
- Azure OpenAI instances: one
- Subscriptions: one
- Regions: one

Topology use cases for multiple model deployments

A topology that includes a single Azure OpenAI instance but contains more than one concurrently deployed model supports the following use cases:

- Expose different model capabilities, such as gpt-35-turbo, gpt-4, and custom fine-tuned models.
- Expose different model versions, such as 0613, 1106, and custom fine-tuned models to support workload evolution or blue-green deployments.
- Expose different quotas assigned (30,000 Token-Per-Minute (TPM), 60,000 TPM) to support consumption throttling across multiple clients.

Introduce a gateway for multiple model deployments



Introducing a gateway into this topology is primarily meant to abstract clients away from self-selecting a specific model instance among the available deployments on the instance. A gateway allows server-side control to direct a client request to a specific model without needing to redeploy client code or change client configuration.

A gateway is especially beneficial when you don't control the client code. It's also beneficial when deploying client configuration is more complex or risky than deploying changes to a gateway routing configuration. You might change which model a client is pointing to based on a blue-green rollout strategy of your model versions, such as rolling out a new fine-tuned model or going from version X to $X+1$ of the same model.

The gateway can also be used as a single API point of entry that enables the gateway to identify the client. It can then determine which model deployment is used to serve the prompt based on that client's identity or other information in the HTTP request. For example, in a multitenant solution, tenants might be limited to specific throughput, and the implementation of the architecture is a model deployment per tenant with specific quotas. In this case, the routing to the tenant's model would be the responsibility of the gateway based on information in the HTTP request.

Tip

Because API keys and Azure role-based access control (Azure RBAC) are applied at the Azure OpenAI instance level and not the model deployment level, adding a gateway in this scenario lets you shift security to the gateway. The gateway then provides additional segmentation between concurrently deployed models that wouldn't otherwise be possible to control through Azure OpenAI's identity and access management (IAM) or IP firewall.

Using a gateway in this topology allows client-based usage tracking. Unless clients are using distinct Microsoft Entra service principals, the access logs for Azure OpenAI wouldn't be able to distinguish multiple clients. Having a gateway in front of the deployment gives your workload an opportunity to track usage per client across various available model deployments to support chargeback or showback models.

Tips for the multiple model deployments topology

- While the gateway is in a position to completely change which model is being used, such as `gpt-35-turbo` to `gpt-4`, that change would likely be considered a breaking change to the client. Don't let new functional capabilities of the gateway distract from always performing [safe deployment practices](#) for this workload.
- This topology is typically straightforward enough to implement through Azure API Management policy instead of a custom code solution.
- To support native SDKs usage with published Azure OpenAI APIs specifications, maintain API compatibility with the Azure OpenAI API. This situation is a larger concern when your team isn't authoring all of your workload clients' code. When deciding designing the HTTP API for the gateway, consider the benefits of maintaining Azure OpenAI HTTP API compatibility.
- While this topology technically supports pass-through client credentials (access tokens or API key) for the Azure OpenAI instance, strongly consider implementing credential termination and reestablishment. This way the client is authorized at the gateway, and then the gateway is authorized through Azure RBAC to the Azure OpenAI instance.
- If the gateway is designed to use pass-through credentials, make sure clients can't bypass the gateway or any model restrictions based on the client.
- Deploy your gateway in the same region as the Azure OpenAI instance.
- Deploy the gateway into a dedicated resource group in the subscription that is separate from the Azure OpenAI instance. Isolating the subscription from the back ends can help drive an [APIOps ↗](#) approach through separations of concern.

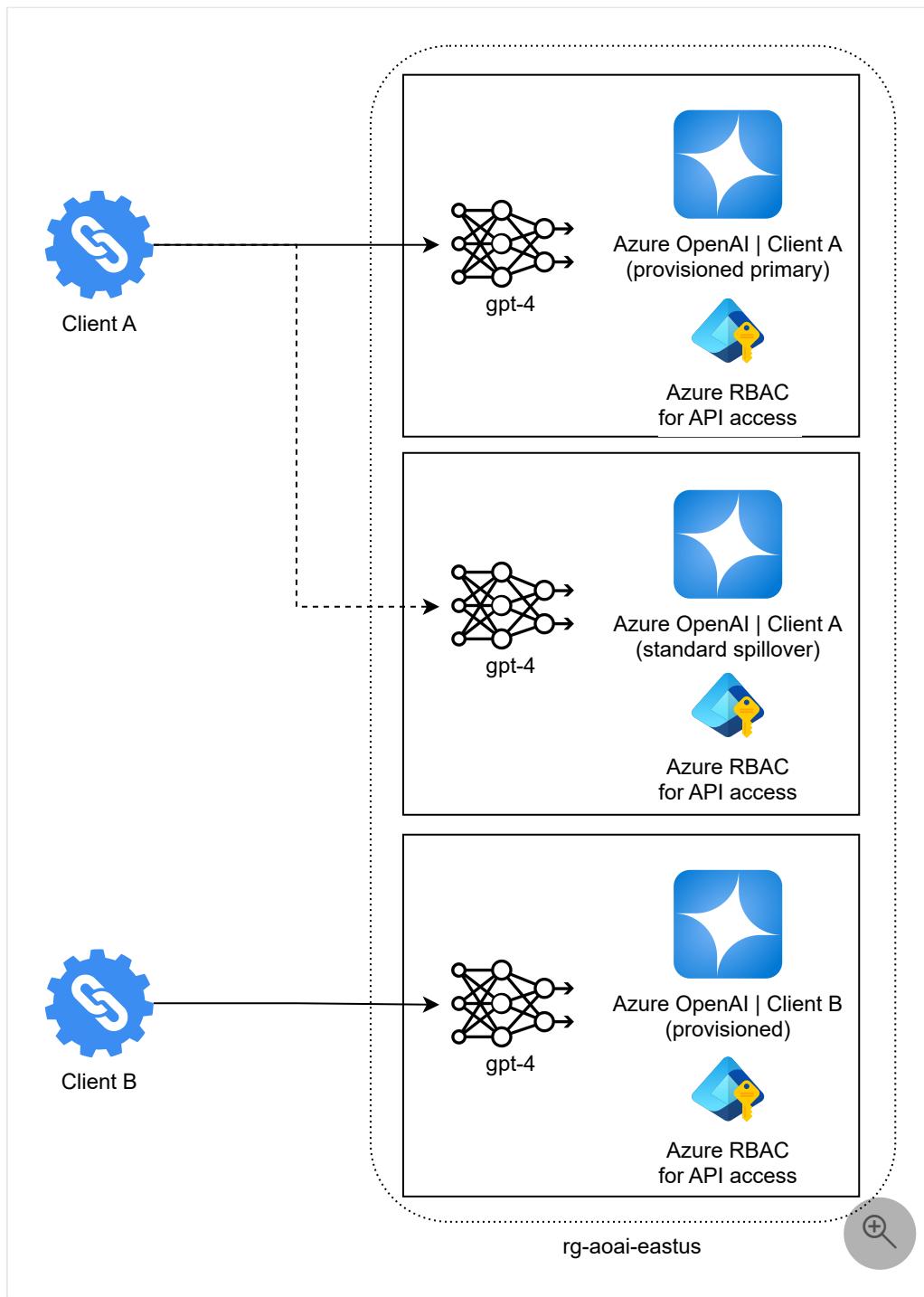
- Deploy the gateway into a virtual network that contains a subnet for the Azure OpenAI instance's Azure Private Link private endpoint. Apply network security group (NSG) rules to that subnet to only allow the gateway access to that private endpoint. All other data plane access to the Azure OpenAI instances should be disallowed.

Reasons to avoid a gateway for multiple model deployments

If controlling your clients' configuration is as easy as or easier than controlling the routing at the gateway level, the added reliability, security, cost, maintenance, and performance impact of the gateway might not be worth the added architectural component.

Also, some workload scenarios could benefit from migrating from a multiple model deployment approach to a multiple Azure OpenAI instance deployment approach. For example, consider multiple Azure OpenAI instances if you have multiple clients that should be using different Azure RBAC or access keys to access their model. Using multiple deployments in a single Azure OpenAI instance and handling logical identity segmentation at the gateway level is possible, but might be excessive when a physical Azure RBAC segmentation approach is available by using distinct Azure OpenAI instances.

Multiple Azure OpenAI instances in a single region and single subscription



Topology details for multiple instances in a single region and single subscription

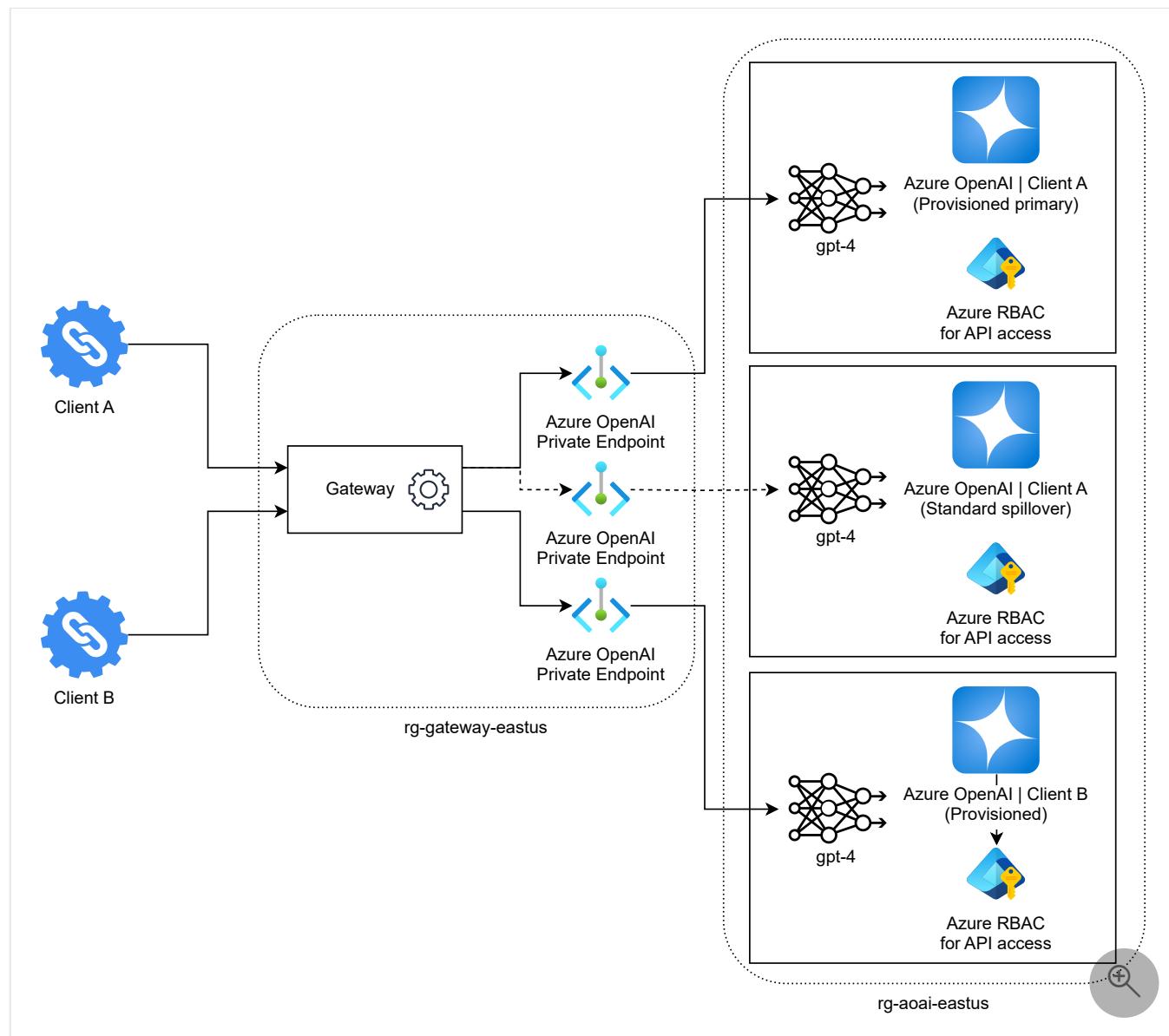
- Azure OpenAI model deployments: one or more
- Azure OpenAI instances: multiple
- Subscriptions: one
- Regions: one

Topology use cases for multiple instances in a single region and single subscription

A topology that includes multiple Azure OpenAI instances in a single region and a single subscription supports the following use cases:

- Enables security segmentation boundaries, such as key or Azure RBAC per client
- Enables an easy chargeback model for different clients
- Enables a failover strategy for Azure OpenAI availability, such as a platform outage that affects a specific instance, a networking misconfiguration, or an accidentally deleted deployment
- Enables a failover strategy for Azure OpenAI quota availability, such as pairing both a provisioned instance and a standard instance for spillover

Introduce a gateway for multiple instances in a single region and subscription



A model might not be accessible to a client for several reasons. These reasons include disruptions in Azure OpenAI Service, Azure OpenAI throttling requests, or issues related to workload operations like network misconfiguration or an inadvertent deletion of a model deployment. To address these challenges, you should implement retry and circuit breaking logic.

This logic could be implemented in clients or server side in a gateway. Implementing the logic in a gateway abstracts the logic away from clients, resulting in no repeated code and a single place to test the logic. No matter if you own the client code or not, this shift can increase reliability of the workload.

Utilizing a gateway with multiple Azure OpenAI instances in a single region and subscription lets you treat all back ends as active-active deployments and not just use them in active-passive failovers. You can deploy the same provisioned model across multiple Azure OpenAI instances and use the gateway to load balance between them.

Note

Standard quotas are subscription level, not Azure OpenAI instance level. Load balancing against standard instances in the same subscription doesn't achieve additional throughput.

One option a workload team has when provisioning Azure OpenAI is deciding if the billing and throughput model is provisioned or standard. A cost optimization strategy to avoid waste through unused provisioned capacity is to slightly under provision the provisioned instance and also deploy a standard instance alongside it. The goal with this topology is to have clients first consume all available pre-allocated throughput and then "burst" over to the standard deployment for overages. This form of planned failover benefits from the same reason as mentioned in the opening paragraph of this section: keeping this complexity out of client code.

When a gateway is involved, it's in a unique position to capture details about all of the model deployments clients are interacting with. While every instance of Azure OpenAI can capture its own telemetry, doing so within the gateway lets the workload team publish telemetry and error responses across all consumed models to a single store, which makes unified dashboarding and alerting easier.

Tips for the multiple instances in a single region and subscription topology

- Ensure the gateway is using the `Retry-After` information available in the HTTP response from Azure OpenAI when supporting failover scenarios at the gateway. Use that

authoritative information to control your circuit-breaker implementation. Don't continuously hit an endpoint that returns a `429 Too Many Requests`. Instead, break the circuit for that model instance.

- Attempting to predict throttling events before they happen by tracking model consumption through prior requests is possible in the gateway, but is fraught with edge cases. In most cases, it's best to not attempt to predict, but use HTTP response codes to drive future routing decisions.
- When round-robinning or failing over to a different endpoint, including provisioned spilling over into standard deployments, always make sure those endpoints are using the same model at the same version. For example, don't fail over from `gpt-4` to `gpt-35-turbo` or from version X to version $X+1$ or load balance between them. This version change can cause unexpected behavior in the clients.
- Load balancing and failover logic are implementable within Azure API Management policies. You might be able to provide a more sophisticated approach using a code-based gateway solution, but API Management is sufficient for this use case.
- Deploy your gateway in the same region as the Azure OpenAI instance.
- Deploy the gateway into a dedicated resource group in the subscription that is separate from the Azure OpenAI instances. Having the gateway isolated from the back ends can help drive an [APIOps](#) approach through separations of concern.
- Colocate all Azure OpenAI instance Private Link private endpoints into a single subnet on the gateway's virtual network. Apply NSG rules to that subnet to only allow the gateway access to those private endpoints. All other data plane access to the Azure OpenAI instances should be disallowed.
- To simplify the logic in your gateway routing code, use the same model deployment name to minimize the difference between the HTTP routes. For example, the model name `gpt4-v1` can be used on all load-balanced or spillover instances, whether it's standard or provisioned.

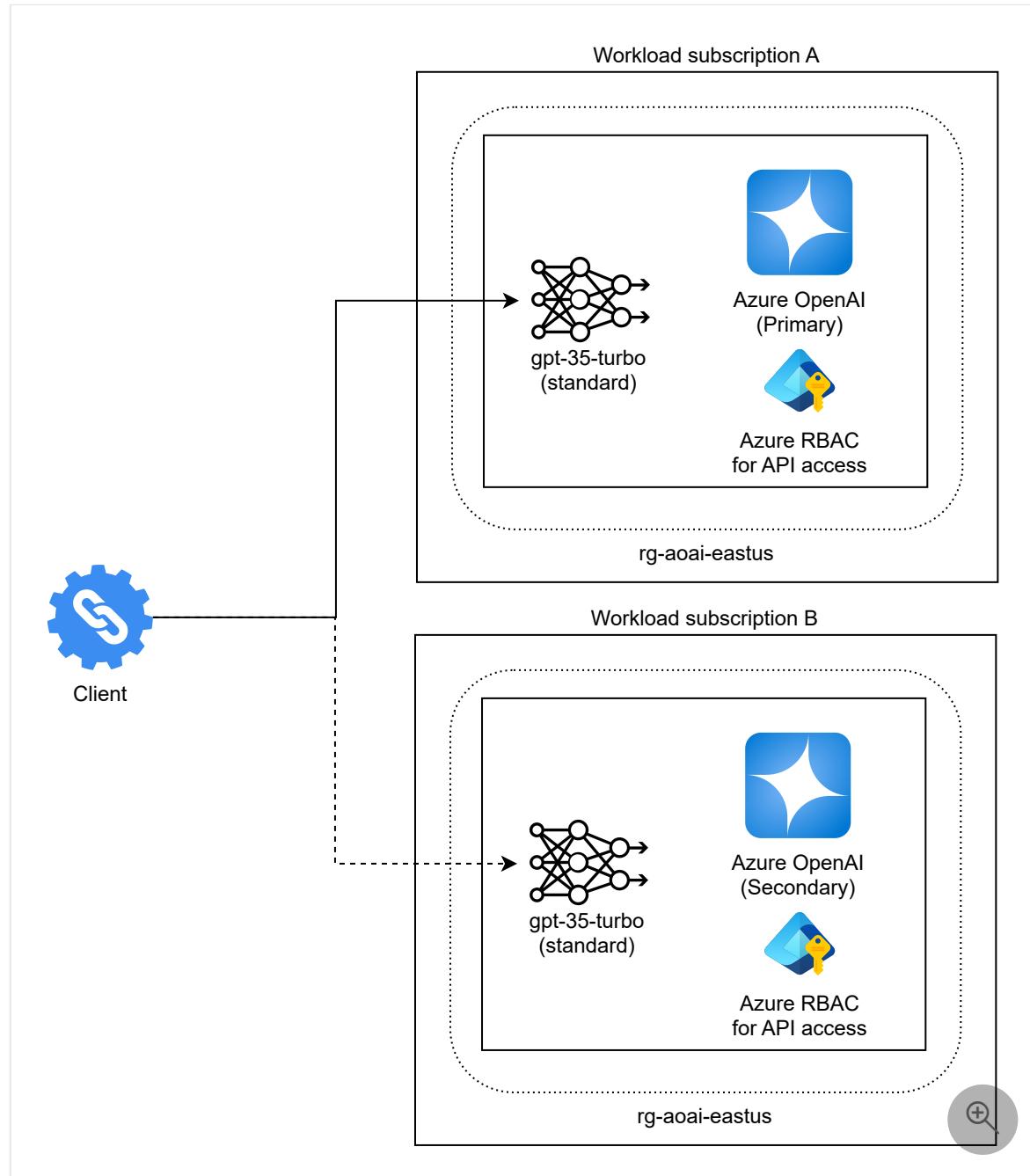
Reasons to avoid a gateway for multiple instances in a single region and subscription

A gateway itself doesn't improve the ability to chargeback models against different clients for this specific topology. In this topology, clients could be granted access to their own dedicated Azure OpenAI instances, which would support your workload team's ability to perform chargeback or showback. This model supports unique identity and network perimeters, so a gateway wouldn't need to be introduced specifically for segmentation.

If you have a few clients in the area where you control the code, and the clients are easily updatable, the logic that you'd have to build into the gateway could be added directly into the code. Consider using the gateway approach for failover or load balancing primarily when you don't own the client code or the complexity is too much for the clients to handle.

If you're using a gateway specifically to address capacity constraints, evaluate if data zone based capacity features are sufficient for your workload.

Multiple Azure OpenAI instances in a single region across multiple subscriptions



Topology details for multiple Azure OpenAI instances in a single region across multiple subscriptions

- Azure OpenAI model deployments: one or more
- Azure OpenAI instances: multiple
- Subscriptions: multiple
- Regions: one

Topology use cases for multiple Azure OpenAI instances in a single region across multiple subscriptions

A topology that includes multiple Azure OpenAI instances in a single region across multiple subscriptions supports the following use cases:

- Includes all of the use cases listed for multiple Azure OpenAI instances in a single region and a single subscription.
- You want to obtain more quota in a standard deployment and you must constrain the use of models to a single, specific region.

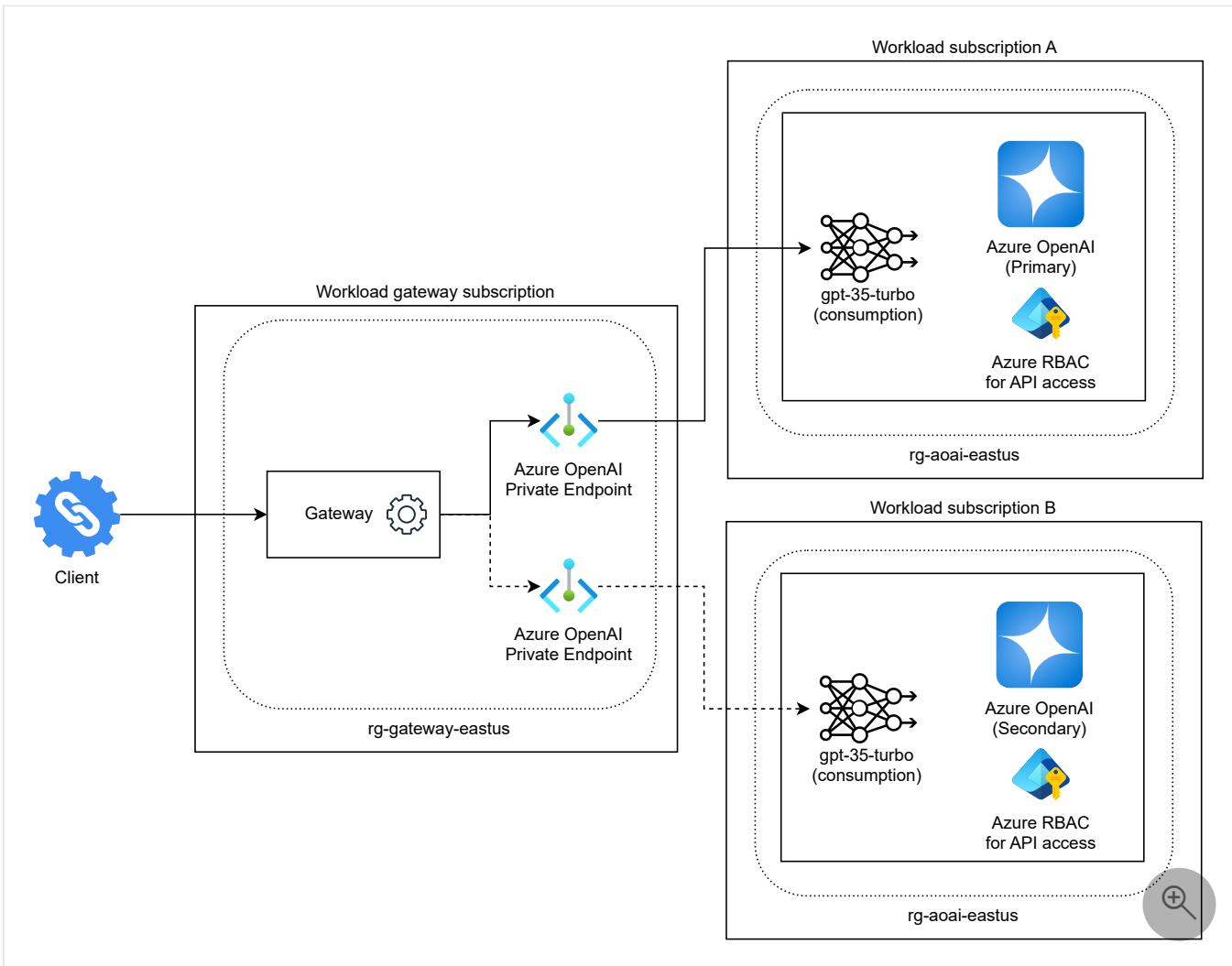
Note

If you don't need to constrain the use of models to a specific region, you should use [global](#) or [data zone](#) deployments of Azure OpenAI that leverage Azure's global infrastructure to dynamically route inferencing requests to data centers with the available capacity.

Introduce a gateway for multiple instances in a single region and multiple subscriptions

The same reasons that are covered in [Introduce a gateway for multiple instances in a single region and subscription](#) apply to this topology.

In addition to those reasons, adding a gateway in this topology also supports a centralized team providing an "Azure OpenAI as a service" model for their organization. Because quota in a standard deployment is subscription-bound, a centralized team that provides Azure OpenAI services that use the standard deployment must deploy Azure OpenAI instances across multiple subscriptions to obtain the required quota. The gateway logic still remains largely the same.



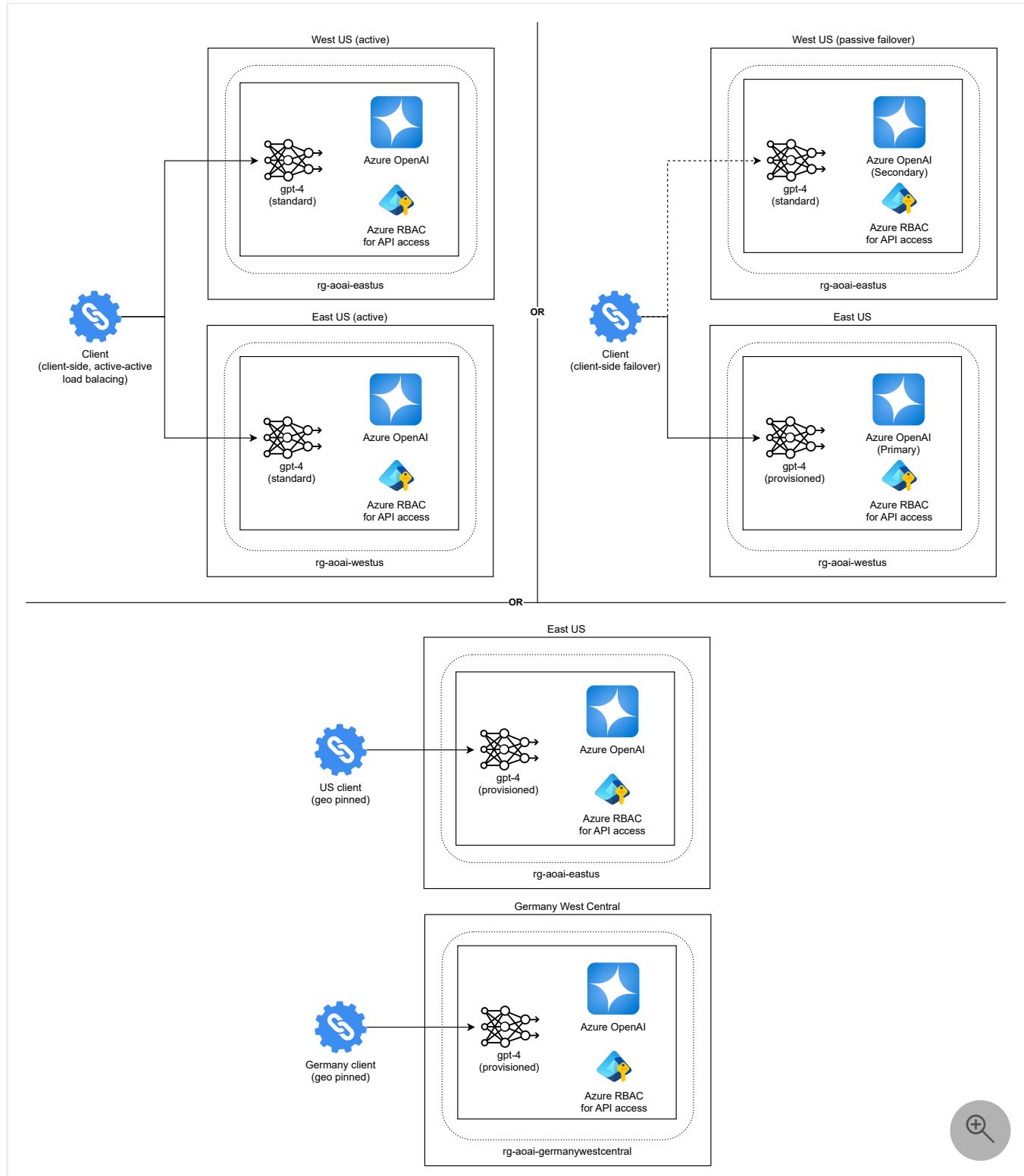
Tips for the multiple instances in a single region and multiple subscriptions topology

- Ideally the subscriptions should all be backed with the same Microsoft Entra tenant to support consistency in Azure RBAC and Azure Policy.
- Deploy your gateway in the same region as the Azure OpenAI instance.
- Deploy the gateway into a dedicated subscription that is separate from the Azure OpenAI instances. This helps enforce a consistency in addressing the Azure OpenAI instances and provides a logical segmentation of duties between Azure OpenAI deployments and their routing.
- When routing requests from your gateway across subscriptions, make sure that private endpoints are reachable. You can use transitive routing through a hub to private endpoints for the back ends in their respective spokes. You might be able to expose private endpoints for the Azure OpenAI services directly in the gateway subscription by using [Private Link connections across subscriptions](#). Cross-subscription Private Link connections are preferred if your architecture and organization support this approach.

Reasons to avoid a gateway for multiple instances in a single region and multiple subscriptions

All of the [reasons to avoid a gateway for multiple instances in a single region and subscription](#) apply to this topology.

Multiple Azure OpenAI instances across multiple regions



Topology details for multiple Azure OpenAI instances across multiple regions

- Azure OpenAI model deployments: multiple
- Azure OpenAI instances: multiple
- Subscriptions: one or more
- Regions: multiple

Topology use cases for multiple Azure OpenAI instances across multiple regions

A topology that includes multiple Azure OpenAI instances spread across two or more Azure regions supports the following use cases:

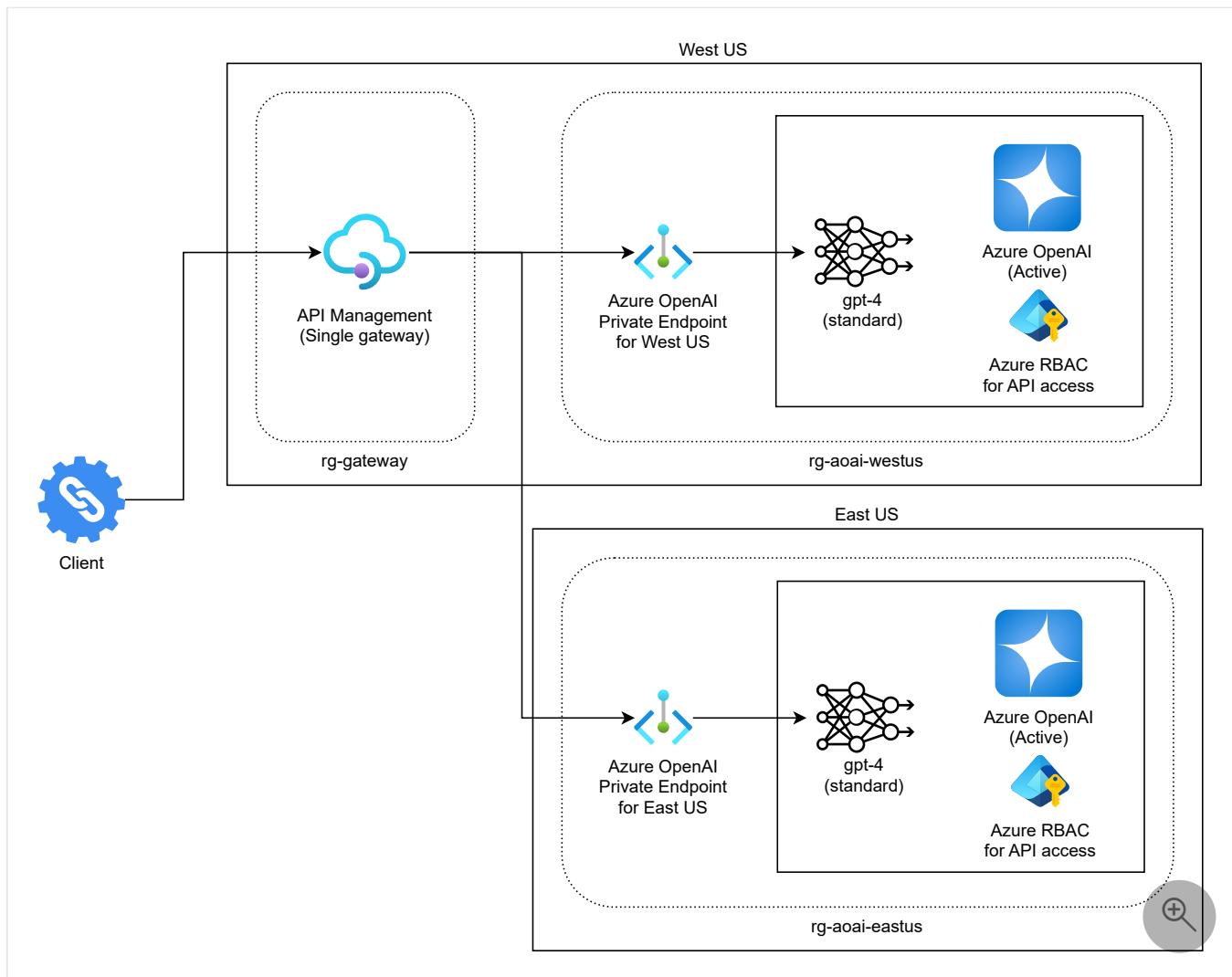
- Includes all of the use cases listed for multiple Azure OpenAI instances in a single region [across multiple subscriptions](#).
- Enables a failover strategy for service availability, such as using [cross-region pairs](#).
- Enables a data residency and compliance design.
- Enables mixed model availability. Some regions have different models and different quotas available for the models.

While technically not different Azure regions, this topology is also applicable when you have an AI model exposed in a cross-premises situation, such as on-premises or in another cloud.

Introduce a gateway for multiple instances in multiple regions

For business-critical architectures that must survive a complete regional outage, a global, unified gateway helps eliminate failover logic from client code. This implementation requires that the gateway remains unaffected by a regional outage.

Use Azure API Management (Single-region deployment)



In this topology, Azure API Management is used specifically for the gateway technology. Here, API Management is deployed into a single region. From that gateway instance, you perform active-active load balancing across regions. The policies in your gateway reference all of your Azure OpenAI instances. The gateway requires network line of sight to each back end across regions, either through cross-region virtual network peering or private endpoints. Calls from this gateway to an Azure OpenAI instance in another region incur more network latency and egress charges.

Your gateway must honor throttling and availability signals from the Azure OpenAI instances and remove faulted back ends from the pool until safe to readd the faulted or throttled Azure OpenAI instance. The gateway should retry the current request against another back-end instance in the pool upon fault, before falling back to returning a gateway error. The gateway's health check should signal unhealthy when no back-end Azure OpenAI instances are available.

! Note

This gateway introduces a global single point of regional failure in your architecture since any service outage on your gateway instances renders all regions inaccessible. Don't use

this topology for business-critical workloads or where client-based load balancing is sufficient.

Because this topology introduces a single point of failure (the gateway), the utility of this specific architecture is fairly limited - protecting you against regional Azure OpenAI endpoint outages.

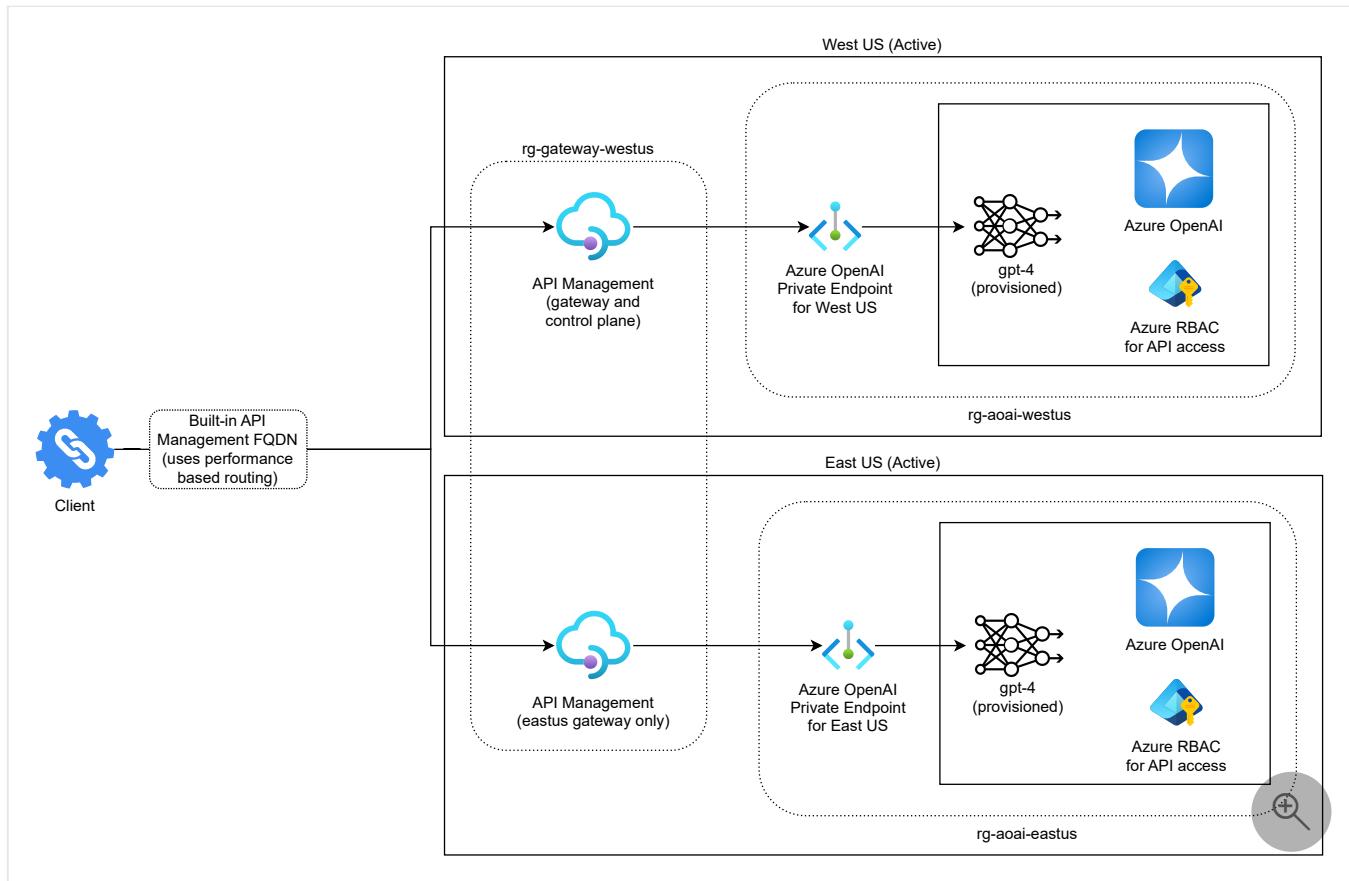
Warning

This approach can't be used in scenarios that involve data sovereignty compliance if either Azure OpenAI region spans a geopolitical boundary.

Active-passive variant

This model can also be used to provide an active-passive approach to specifically handle regional failure of just Azure OpenAI. In this mode, traffic normally flows from the gateway to the Azure OpenAI instance in the same region as the API management service. That instance of Azure OpenAI would handle all expected traffic flow without regional failures occurring. It could be provisioned or standard, depending on your preferred billing model. In the case of a regional failure of just Azure OpenAI, the gateway can redirect traffic to another region with Azure OpenAI already deployed in a standard deployment.

Use Azure API Management (Multi-region deployment)



To improve the reliability of the prior Azure API Management-based architecture, API Management supports deploying an [instance to multiple Azure regions](#). This deployment option gives you a single control plane, through a single API Management instance, but provides replicated gateways in the regions of your choice. In this topology, you deploy gateway components into each region containing Azure OpenAI instances that provide an active-active gateway architecture.

Policies such as routing and request handling logic are replicated to each individual gateway. All policy logic must have conditional logic in the policy to ensure that you're calling Azure OpenAI instances in the same region as the current gateway. For more information, see [Route API calls to regional back end services](#). The gateway component then requires network line of sight only to Azure OpenAI instances in its own region, usually through private endpoints.

! Note

This topology doesn't have a global point of failure of a traffic handling perspective, but the architecture partially suffers from a single point of failure in that the Azure API Management control plane is only in a single region. Evaluate whether the control plane limitation might violate your business or mission-critical standards.

API Management offers out-of-the-box global fully qualified domain name (FQDN) routing based on lowest latency. Use this built-in, performance based functionality for active-active gateway deployments. This built-in functionality helps address performance and handles a

regional gateway outage. The built-in global router also supports disaster recovery testing as regions can be simulated down through disabling individual gateways. Make sure clients respect the time to live (TTL) on the FQDN and have appropriate retry logic to handle a recent DNS failover.

If you need to introduce a web application firewall into this architecture, you can still use the built-in FQDN routing solution as the back-end origin for your global router that implements a web application firewall. The global router would delegate failover responsibility to API Management. Alternatively, you could use the regional gateway FQDNs as the back-end pool members. In that latter architecture, use the built-in `/status-0123456789abcdef` endpoint on each regional gateway or another custom health API endpoint to support regional failover. If unsure, start with the single origin back-end FQDN approach.

This architecture works best if you can treat regions as either fully available or fully unavailable. This means that if either the API Management gateway or Azure OpenAI instance is unavailable, you want client traffic to no longer be routed to the API Management gateway in that region. Unless another provision is made, if the regional gateway still accepts traffic while Azure OpenAI is unavailable, the error must be propagated to the client. To avoid the client error, see an improved approach in [Active-active gateway plus active-passive Azure OpenAI variant](#).

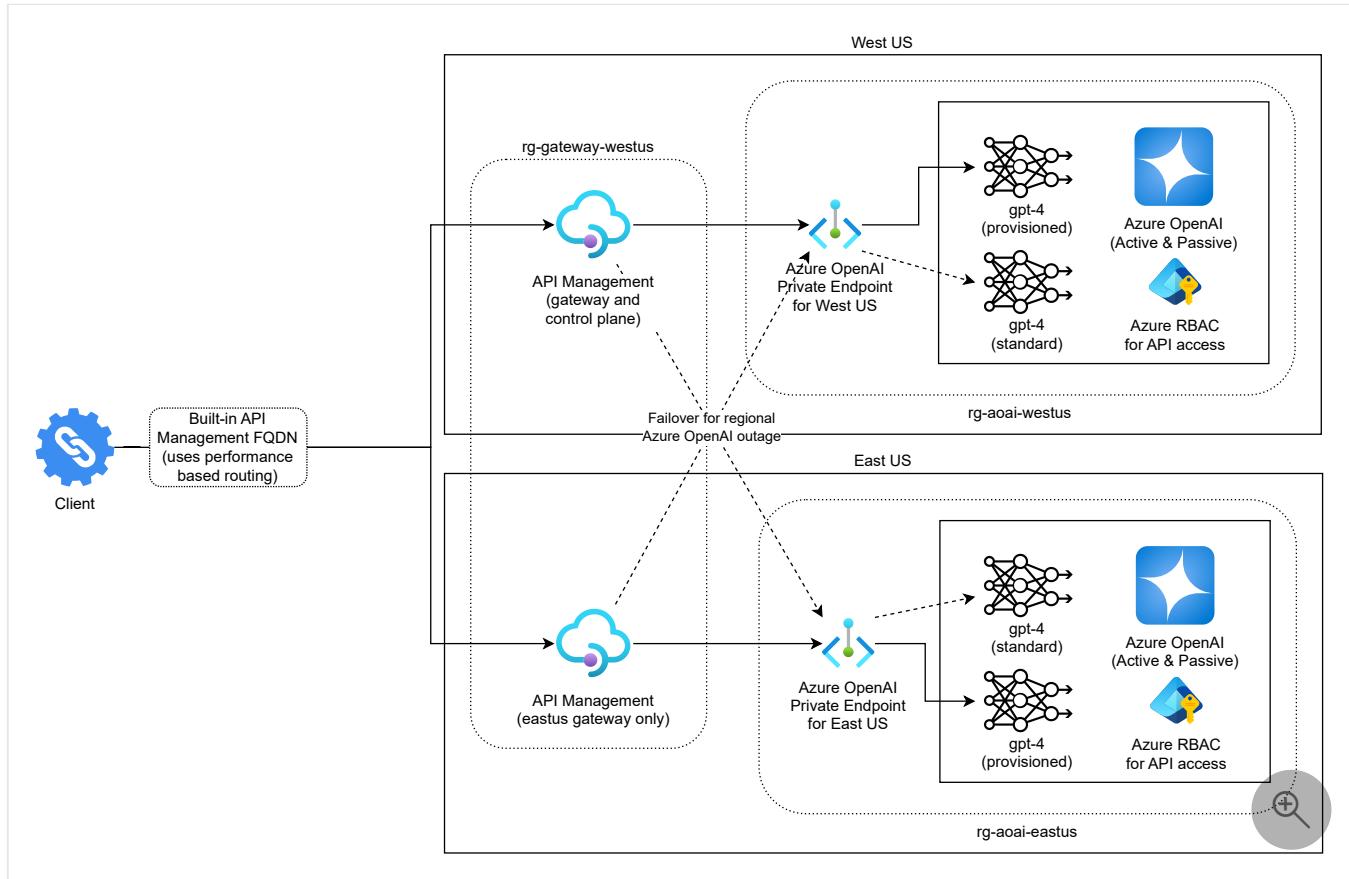
If a region is experiencing an API Management gateway outage or is flagged as unhealthy, the remaining available regions need to absorb 100% of the traffic from those other regions. This means you need to over-provision provisioned Azure OpenAI instances to handle the new burst of traffic or use an [active-passive approach for failover](#). Use the [Azure OpenAI Capacity calculator](#) for capacity planning.

Ensure that the resource group that contains Azure API Management is the same location as the API Management instance itself to reduce the blast radius of a related regional outage affecting your ability to access the resource provider for your gateways.

Warning

This approach can't be used in scenarios that involve data residency compliance if either gateway region spans a geopolitical boundary.

[Active-active gateway plus active-passive Azure OpenAI variant](#)

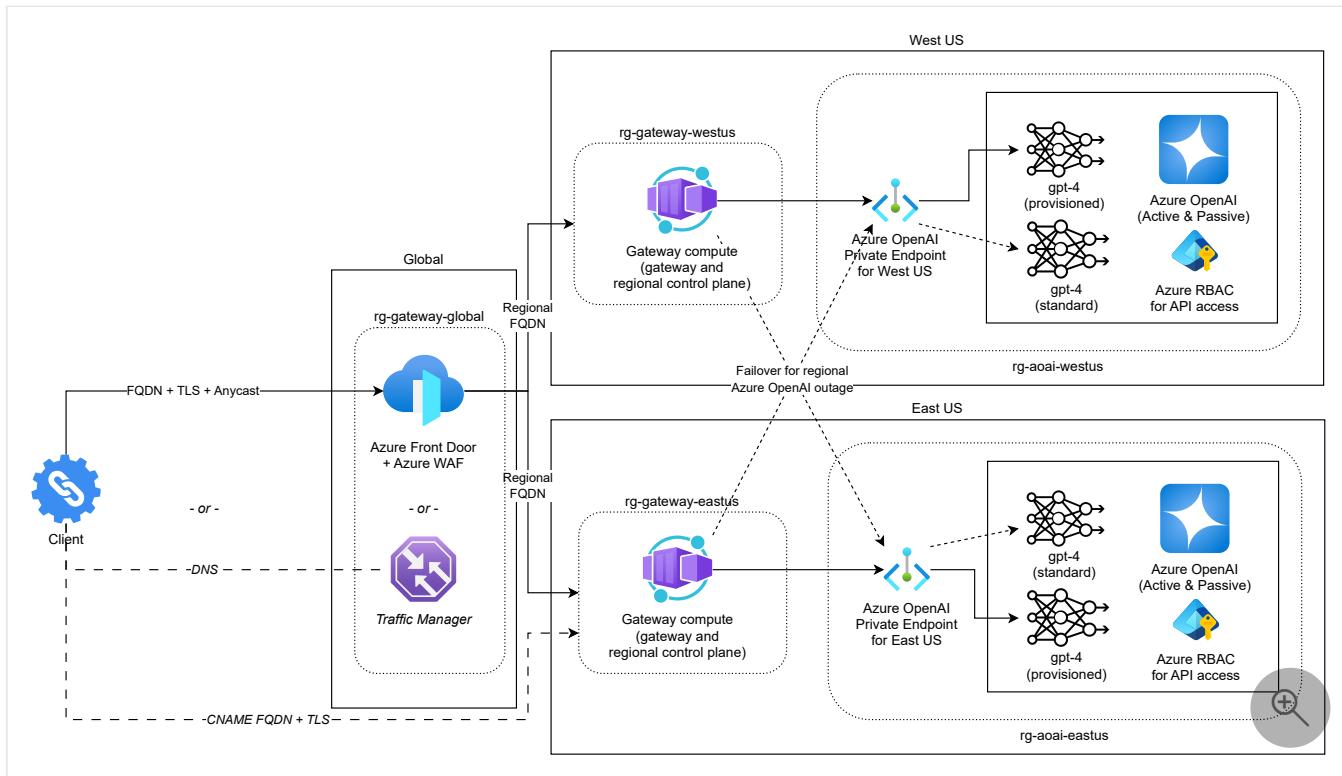


The previous section addresses the availability of the gateway by providing an active-active gateway topology. This topology combines the active-active gateway with a cost-effective active-passive Azure OpenAI topology. Adding active-passive logic to the gateway to fail over to a standard Azure OpenAI deployment from a provisioned deployment can significantly increase the reliability of the workload. This model still allows clients to use the API Management built-in FQDN routing solution for performance based routing.

⚠️ Warning

This active-active plus active-passive approach can't be used in scenarios that involve data residency compliance if either region spans a geopolitical boundary.

Use a custom coded gateway



If your per-gateway routing rules are too complex for your team to consider reasonable as Azure API Management policies, you need to deploy and manage your own solution. This architecture must be a multi-region deployment of your gateway, with one highly available scale unit per region. You need to front those deployments with [Azure Front Door](#) (Anycast) or [Azure Traffic Manager](#) (DNS), typically by using latency-based routing and appropriate health checks of gateway availability.

Use Azure Front Door if you require a web application firewall and public internet access. Use Traffic Manager if you don't need a web application firewall and DNS TTL is sufficient. When fronting your gateway instances with Azure Front Door (or any reverse proxy), ensure that the gateway can't be bypassed. Make the gateway instances available only through private endpoint when you use Azure Front Door and add validation of the `x_AZURE_FDID` HTTP header in your gateway implementation.

Place per-region resources that are used in your custom gateway in per-region resource groups. Doing so reduces the blast radius of a related regional outage affecting your ability to access the resource provider for your gateway resources in that region.

You can also consider fronting your gateway logic implementation with Azure API Management, for the other benefits of API Management, such as TLS, authentication, health check, or round-robin load balancing. Doing so shifts common API concerns out of custom code in your gateway and lets your gateway specifically address Azure OpenAI instance and model deployment routing.

For data residency compliance, make sure each geopolitical boundary has its own isolated deployment of this architecture and that clients can only reach their authorized endpoint.

Reasons to avoid a gateway for multiple instances in multiple regions

Don't implement a unified gateway across geopolitical regions when data residency and compliance is required. Doing so would violate the data residency requirements. Use individually addressable gateways per region, and follow the guidance in one of the previous sections.

Don't implement a unified gateway solely for the purpose of increasing quota. Use [Global Standard](#) deployments of Azure OpenAI leverage Azure's global infrastructure to dynamically route requests to data centers with the best capacity for each request.

If clients aren't expected to fail over between regions and you have the ability to give clients a specific gateway to use, then instead use multiple gateways, one per region, and follow the guidance in one of the previous sections. Don't tie the availability of other regions to the region containing your gateway as a single point of failure.

Don't implement a unified gateway if your model and version isn't available in all regions exposed by the gateway. Clients need to be routed to the same model and the same model version. For multi-region load-balanced and failover gateways, you need to pick a common model and model version that is available across all involved regions. For more information, see [Model availability](#). If you can't standardize on model and model version, the benefit of the gateway is limited.

General recommendations

No matter which topology your workload needs, there are a few cross-cutting recommendations to consider when building your gateway solution.

Stateful interactions

When clients use Azure OpenAI's stateful features, such as the Assistants API, you need to configure your gateway to pin a client to a specific back end during that interaction. Doing so can be accomplished by storing instance data in a cookie. In these scenarios, consider returning an API response like a `429` to a pinned client instead of redirecting them to a different Azure OpenAI instance. Doing so allows the client to explicitly handle sudden unavailability versus hiding it and being routed to a model instance that has no history.

Gateway health checks

There are two health check perspectives to consider, regardless of topology.

If your gateway is built around round-robinning or strictly performing service availability failover, you want a way to take an Azure OpenAI instance (or model) out of availability status. Azure OpenAI doesn't provide any sort of health check endpoint to preemptively know whether it's available to handle requests. You could send synthetic transitions through, but that consumes model capacity. Unless you have another reliable signal source for Azure OpenAI instance and model availability, your gateway likely should assume the Azure OpenAI instance is up and then handle `429`, `500`, `503` HTTP status codes as a signal to circuit-break for future requests on that instance or model for some time. For throttling situations, always honor the data in the `Retry-After` header found in Azure OpenAI API responses for `429` response codes in your circuit breaking logic. If you're using Azure API Management, evaluate using the [built-in circuit breaker](#) functionality.

Your clients or your workload operations team might wish to have a health check exposed on your gateway for their own routing or introspection purposes. If you use API Management, the default `/status-0123456789abcdef` might not be detailed enough since it mostly addresses the API Management gateway instance, not your back ends. Consider adding a dedicated health check API that can return meaningful data to clients or observability systems on the availability of the gateway or specific routes in the gateway.

Safe deployment practices

You can use gateway implementations to orchestrate blue-green deployments of updated models. Azure OpenAI models are updated with new model versions and new models, and you might have new fine-tuned models.

After testing the effects of a change in preproduction, evaluate whether production clients should be "cut over" to the new model version or instead shift traffic. The gateway pattern described earlier allows the back end to have both models concurrently deployed. Deploying models concurrently gives the power to the gateway to redirect traffic based on the workload team's safe deployment practice of incremental rollout.

Even if you don't use blue-green deployments, your workload's APIOps approach needs to be defined and sufficiently automated commiserate with the rate of change of your back-end instance and model deployments.

Just enough implementation

Many of the scenarios introduced in this article help increase the potential service-level objective (SLO) of your workload by reducing client complexity and implementing reliable self-preservation techniques. Others improve the security of the workload by moving access controls to specific models away from Azure OpenAI. Be sure that the introduction of the

gateway doesn't end up working counter to these goals. Understand the risks of adding a new single point of failure either through service faults or human-caused configuration issues in the gateway, complex routing logic, or the risks of exposing more models to unauthorized clients than is intended.

Data sovereignty

The various active-active and active-passive approaches need to be evaluated from a data residency compliance perspective for your workload. Many of these patterns would be applicable for your workload's architecture if the regions involved remain within the geopolitical boundary. To support this scenario, you need to treat geopolitical boundaries as isolated stamps and apply the active-active or active-passive handling exclusively within that stamp.

In particular, any performance-based routing needs to be highly scrutinized for data sovereignty compliance. In data sovereignty scenarios, you can't service clients with another geography and remain compliant. All gateway architectures that involve data residency must enforce that clients only use endpoints in their geopolitical region. The clients must be blocked from using other gateway endpoints and the gateway itself doesn't violate the client's trust by making a cross-geopolitical request. The most reliable way to implement this segmentation is to build your architecture around a fully independent, highly available gateway per geopolitical region.

When considering whether to take advantage of increased capacity through [global](#) or [data zone](#) deployments of Azure OpenAI, you need to understand how these deployments affect data residency. Data stored at rest remains in the designated Azure geography for both global and data zone deployments. That data may be transmitted and processed for inferencing in any Azure OpenAI location for global deployments, or in any Azure OpenAI location within the Microsoft specified data zone for data zone deployments.

Azure OpenAI authorization

The gateway needs to authenticate with all Azure OpenAI instances that it interfaces with. Unless you designed the gateway to do pass-through authentication, the gateway should use a managed identity for its credentials. So each Azure OpenAI instance needs to configure least-privileged Azure RBAC for the gateways' managed identities. For active-active and failover architectures, make sure the gateway's identity has equivalent permissions across all involved Azure OpenAI instances.

Azure Policy

Consistency between model deployments and Azure OpenAI instances is important in both active-active and active-passive situations. Use Azure Policy to help enforce consistency between the various Azure OpenAI instances or model deployments. If the [built-in policies](#) for Azure OpenAI aren't sufficient to ensure consistency between them, consider creating or using [community created](#) custom policies.

Gateway redundancy

While not specific to multiple back ends, each region's gateway implementation should always be built with redundancy and be highly available within the scale unit. Choose regions with availability zones and make sure your gateway is spread across them. Deploy multiple instances of the gateway so that single point of failure is limited to a complete regional outage and not the fault of a single compute instance in your gateway. For Azure API Management, deploy two or more units across two or more zones. For custom code implementations, deploy at least three instances with best effort distribution across availability zones.

Gateway implementations

Azure doesn't offer a complete turn-key solution or reference architecture for building such a gateway that focuses on routing traffic across multiple backends. However, Azure API Management is preferred as the service gives you a PaaS based solution using built in features such as backend pools, circuit-breaking policies, and custom policies if needed. See, [Overview of generative AI gateway capabilities in Azure API Management](#) to evaluate what is available in that service for your workload's multi-backend needs.

Whether you use API Management or build a custom solution, as mentioned in the [introduction article](#), your workload team must build and operate this gateway. Following are examples covering some of the previously mentioned use cases. Consider referencing these samples when you build your own proof of concept with API Management or custom code.

[] [Expand table](#)

| Implementation | Example |
|----------------------|--|
| Azure API Management | Smart load balancing for Azure OpenAI using Azure API Management - This GitHub repo contains sample policy code and instructions to deploy into your subscription. |
| | Scaling Azure OpenAI using Azure API Management - This GitHub repo contains sample policy code and instructions for provisioned and standard spillover. |

| Implementation | Example |
|----------------|---|
| | The GenAI gateway toolkit contains example API Management policies along with a load-testing setup for testing the behavior of the policies. |
| Custom code | Smart load balancing for Azure OpenAI using Azure Container Apps This GitHub repo contains sample C# code and instructions to build the container and deploy into your subscription. |

The Cloud Adoption Framework for Azure also contains guidance on implementing an [Azure API Management landing zone](#) for generative AI scenarios, including this multi-backend scenario. If your workload exists in an application landing zone, be sure to refer to this guidance for implementation considerations and recommendations.

Next steps

Having a gateway implementation for your workload provides benefits beyond the tactical multiple back end routing benefit described in this article. Learn about the other [key challenges](#) a gateway can solve.

Related resources

- [Design a well-architected AI workload](#)
- [API gateway in Azure API Management](#)

Provide custom authentication to Azure OpenAI Service through a gateway

Azure AI services

Azure OpenAI Service

Azure API Management

Microsoft Entra ID

Intelligent applications that use Azure OpenAI Service through Azure-native services provide user authentication and authorization. However, some scenarios are complex and require different architecture designs. These scenarios include topologies that have client applications that aren't hosted on Azure, use external identity providers, and deploy multiple clients that access the same Azure OpenAI instances. In these scenarios, introducing a gateway in front of Azure OpenAI can significantly improve security by adding a layer that ensures consistent authentication to deployed instances.

This article describes key scenarios that provide authentication to Azure OpenAI:

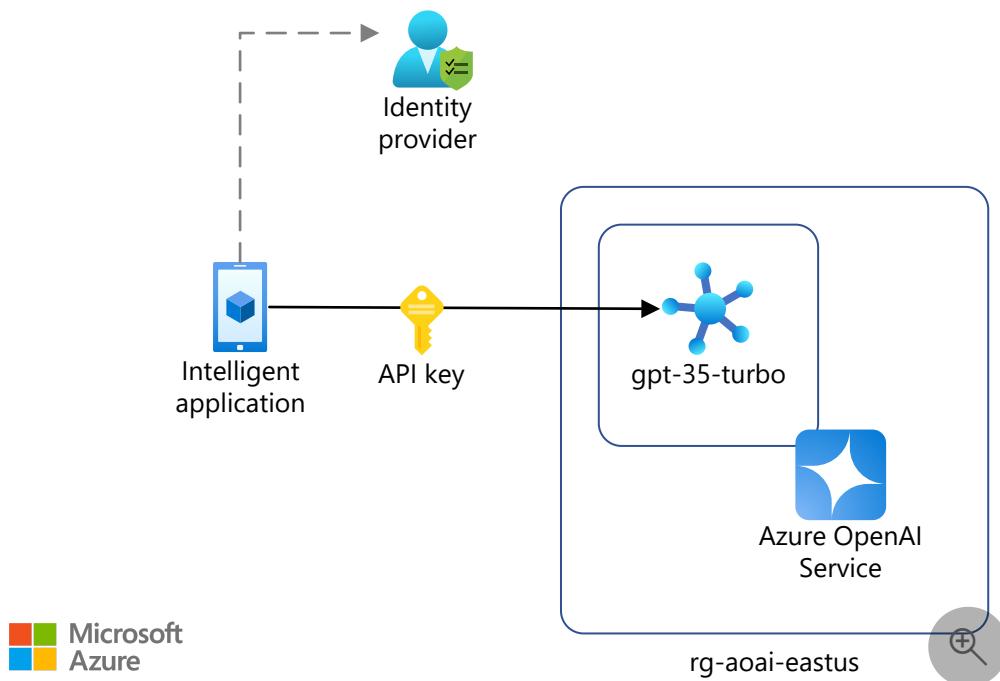
- [Authenticate client applications via an external identity provider](#)
- [Authenticate client applications via certificates](#)
- [Authenticate multiple client applications via keys to access a shared Azure OpenAI instance](#)
- [Authenticate client applications that access multiple Azure OpenAI instances](#)

Each scenario describes the challenges that they introduce and the benefits of incorporating a gateway.

Important

You can use the following guidance for any gateway implementation, including Azure API Management. To illustrate this flexibility, the architecture diagrams use generic representations of components in most scenarios.

Authenticate client applications via an external identity provider



[Download a *Visio file*](#) of this architecture.

Scenario constraints

This scenario has the following constraints:

- Client applications use an external OpenID Connect (OIDC)-enabled identity provider such as Okta, Auth0, or social identity providers.
- Client applications authenticate against a Microsoft Entra tenant that's different than the Azure OpenAI data plane's tenant.

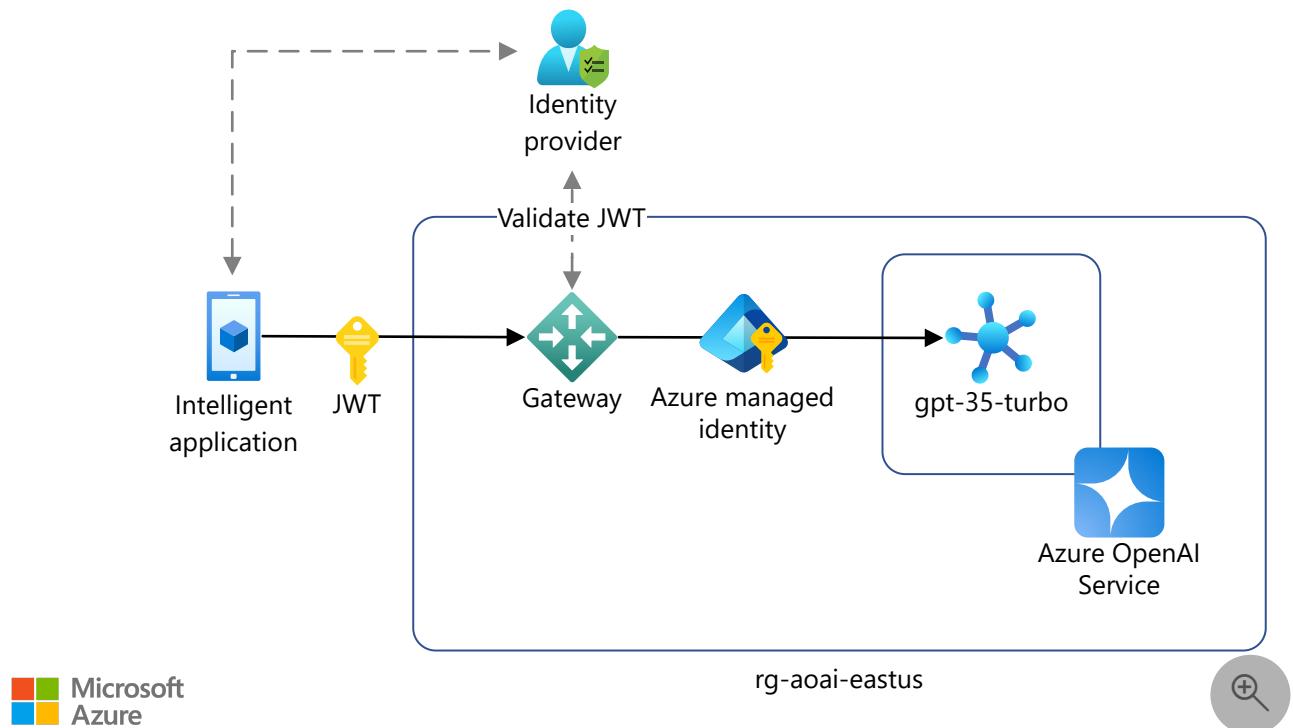
These constraints can apply to the following examples:

- Existing client applications that already authenticate against an external OIDC provider or Microsoft Entra ID and that integrate with Azure OpenAI instances.
- Client applications that must consistently authenticate users from multiple identity providers.

Connect directly to Azure OpenAI

If the client applications in these scenarios directly connect to Azure OpenAI without using a gateway, they must use key-based authentication to authenticate to Azure OpenAI. Key-based authentication introduces extra security concerns. You must securely store and rotate keys, and you can't give different clients Azure role-based access control (Azure RBAC) configurations for individual model deployments.

Introduce a gateway



Download a [Visio file](#) of this architecture.

A gateway resolves this scenario's challenges in the following ways:

- The gateway uses Open Authorization (OAuth) to authenticate users via their existing external identity providers. The gateway validates the authenticated user access tokens, such as a JSON Web Token (JWT), that the identity provider generates. Then the gateway grants authorization to the backing Azure OpenAI instance.
- You don't need to manage client keys. This approach eliminates the security risks of key-based authentication.
- The gateway connects to Azure OpenAI by using a managed identity, which improves security via least-privileged Azure RBAC.

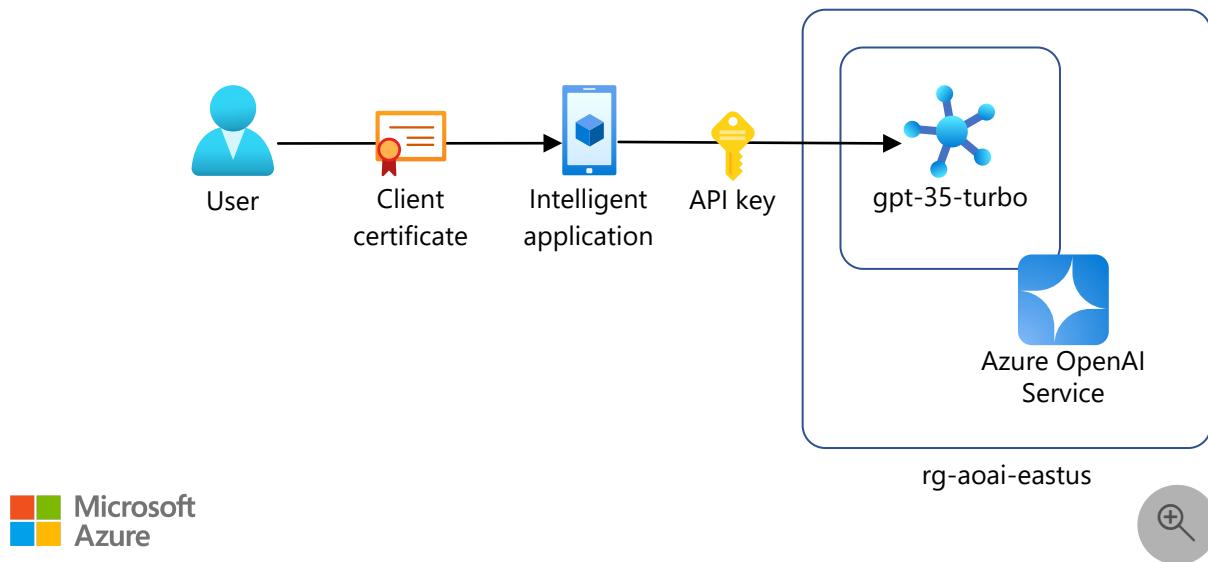
Recommendations for this scenario

- Add more OAuth scopes to your application registration in your identity provider to enable granular permission to consumers. These scopes enable client applications to request permission to perform specific operations in your gateway, including [access to Azure OpenAI](#).
- Configure this scenario for API Management by using inbound policies. Use the [validate-jwt policy](#) to enforce the existence, validity, and attribute values of a supported JWT.

Reasons to avoid a gateway for this scenario

If a single intelligent application accesses Azure OpenAI, it's easier to configure user authentication and authorization within the app rather than through the gateway. Use this approach to assign the necessary Azure RBAC to securely authenticate the intelligent application with Azure OpenAI.

Authenticate client applications via certificates



Download a [Visio file](#) of this architecture.

Scenario constraints

This scenario has the following constraints:

- You want to use certificates to authenticate client applications.
- Client applications can't use, or you don't want to use, Microsoft Entra ID or other OIDC providers for authentication.
- Clients can't use, or you don't want to use, federated identity for authentication.

These constraints can apply to the following examples:

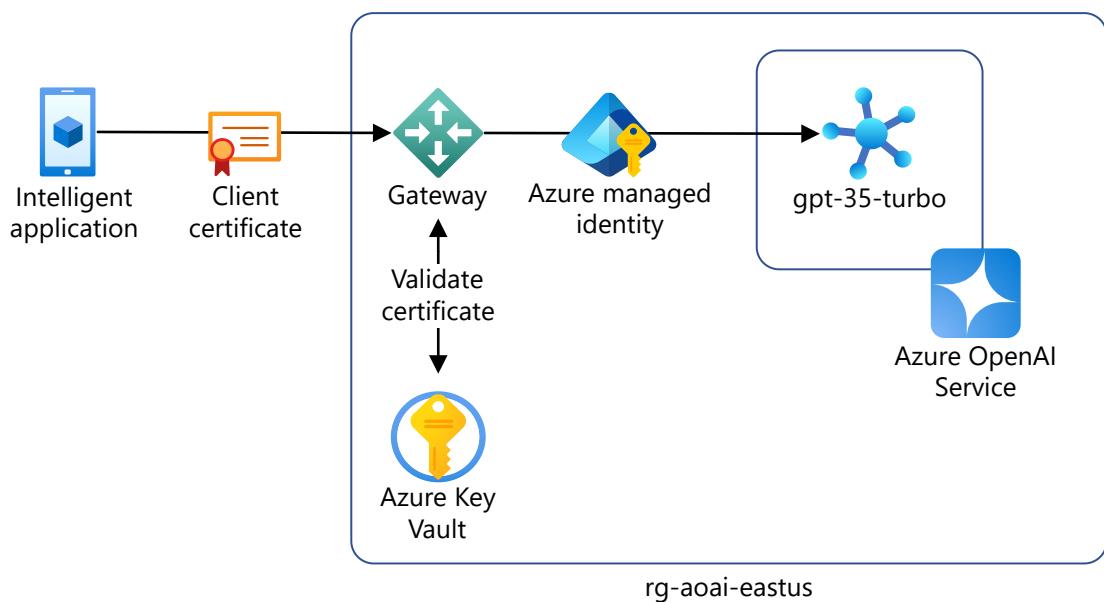
- A client that authenticates to Azure OpenAI is a machine or device and no user interaction occurs.
- Your organization requires that you use certificates for authentication because of security standards and compliance regulations.

- You want to provide multiple client applications with options to authenticate based on their environment, including using client certificates.

Connect directly to Azure OpenAI

Azure OpenAI doesn't natively support client certification authentication. To support this scenario without a gateway, the intelligent application needs to use certificate authentication for the user and either an API key or managed identity to authenticate requests to the Azure OpenAI instance. You must implement the certificate authentication logic in every client. In this scenario, key-based authentication introduces risks and management overhead if you connect directly to Azure OpenAI from clients.

Introduce a gateway



Download a [Visio file](#) of this architecture.

You can introduce a gateway into this architecture that offloads client certification validation from the clients. The gateway [validates the client digital certificate](#) that the intelligent application presents and checks the issuer, expiration date, thumbprint, and revocation lists. The gateway should use a managed identity to authenticate itself with Azure OpenAI. The gateway should also use Azure Key Vault to store the root certificate authority (CA). Use this approach to centralize client certificate validation, which reduces maintenance overhead.

A gateway provides several advantages in this scenario:

- You use the managed identity of the gateway instead of access keys, which eliminates the risk stolen keys and reduces the maintenance burden of key rotation.
- You can centralize certificate validation, which ensures that you use consistent security policies to evaluate client digital certificates for all intelligent applications.
- You can offload certificate validation to the gateway, which simplifies client code.

Recommendations for this scenario

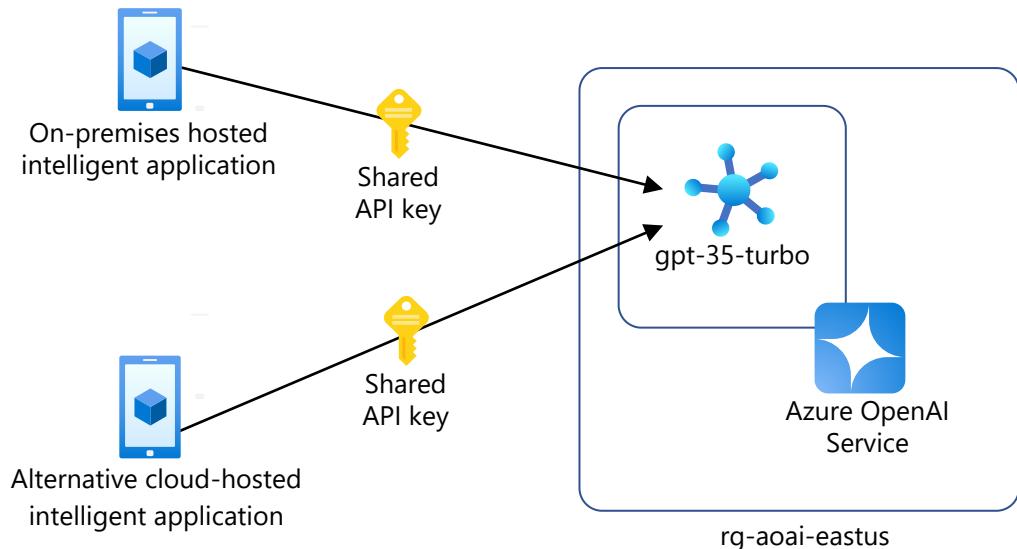
- Verify the entire certificate chain, including the root CA and intermediate certificates, when you validate certificates. Full verification ensures the authenticity of the certificate and prevents unauthorized access.
- Rotate and renew client certificates regularly to minimize the risk of certificate compromise. Use Key Vault to automate this process and keep certificates up to date. Set alerts for upcoming certificate expirations to prevent service disruptions at the gateway.
- Implement mutual Transport Layer Security (mTLS) to ensure that both the client and server authenticate each other. This strategy provides an extra layer of security. To configure the gateway to enforce mTLS, set the appropriate policies and constraints.
- Use the [validate-client-certificate policy](#) in API Management to validate client certificates that an Azure Key Vault references. This policy validates the client certificate that the client application presents and checks the issuer, expiration date, thumbprint, and revocation lists.

Reasons to avoid a gateway for this scenario

In simple environments that have few clients, the cost of handling security and certificate management in the client can outweigh the added complexity of introducing a gateway. Also, gateways can become single points of failure, increase latency because of added layers, and lead to vendor lock-in if you choose commercial solutions rather than custom implementations.

You must carefully assess your specific needs, resource availability, and the criticality of your applications before you implement a gateway for client certificate authentication.

Authenticate multiple client applications via keys to access a shared Azure OpenAI instance



Download a [Visio file](#) of this architecture.

Scenario constraints

This scenario has the following constraints:

- Multiple client applications access a shared Azure OpenAI instance.
- Clients can't use, or you don't want to use, Microsoft Entra ID for authentication.
- Clients can't use, or you don't want to use, federated identity for authentication.
- You want to use key-based authentication for client applications.

These constraints can apply to the following examples:

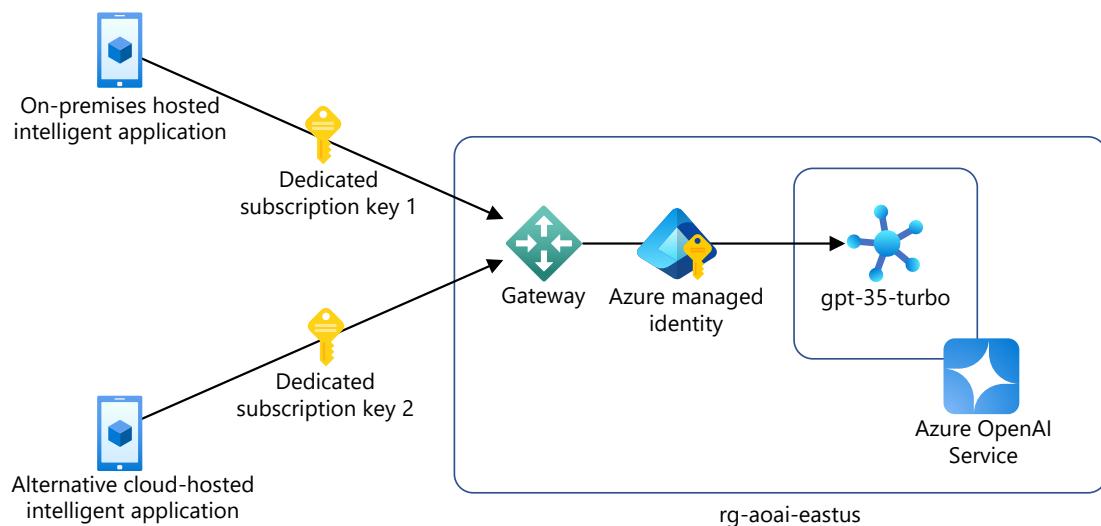
- You deploy client applications across multiple environments, including Azure, on-premises, or other cloud providers.
- An organization needs to provide Azure OpenAI to different teams and set unique access and usage limits for each team.

Connect directly to Azure OpenAI

Azure OpenAI supports key-based authentication via shared keys. Azure OpenAI exposes a primary key and a secondary key. The purpose of the secondary key is to support key rotation. It doesn't provide client identity isolation. When you authenticate multiple clients directly to Azure OpenAI in this scenario, each client shares the same key. This implementation has the following challenges:

- You can't revoke permissions for specific clients because every client shares the same key.
- You can't give different clients different access rights to different models in the same Azure OpenAI instance deployment.
- You can't differentiate one client from another from a logging perspective.

Introduce a gateway



Download a [Visio file](#) of this architecture.

You can introduce a gateway into this architecture that issues a dedicated key to each client application. API Management uses the concept of [subscriptions](#) to provide dedicated client keys. The gateway should use a managed identity to authenticate itself with Azure OpenAI.

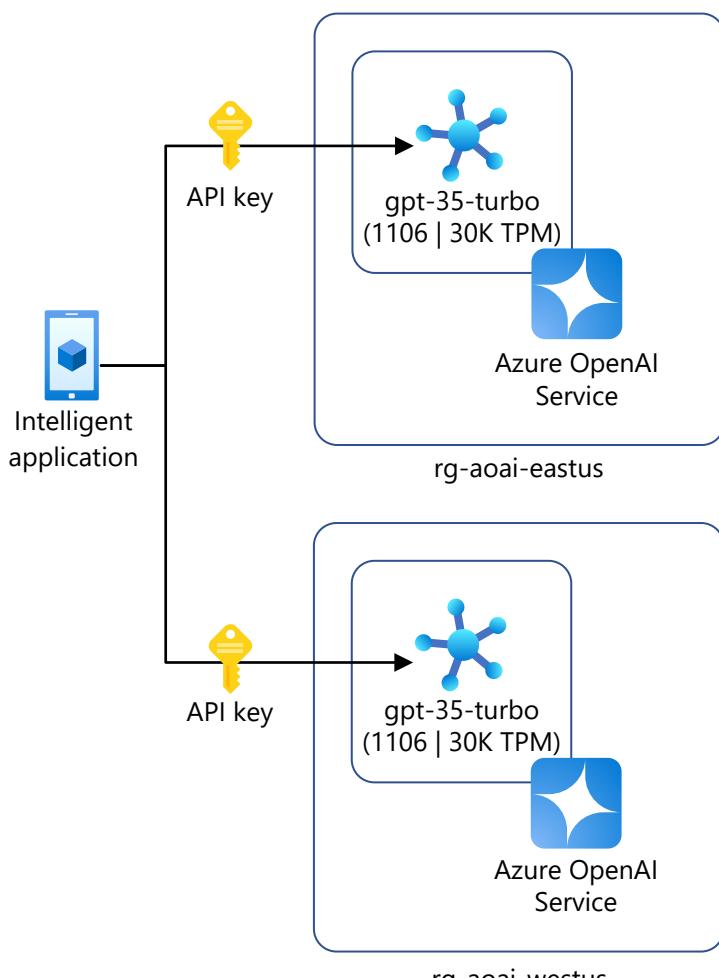
A gateway provides several advantages in this scenario:

- You can revoke access to a single client application without affecting other clients.
- You don't need to update all the client's key configurations before you rotate keys, so key rotation is logically easier. You can rotate the dedicated keys for each client after you update the client configuration.
- You can uniquely identify each client from a logging perspective.
- The gateway enforces rate limits and quotas for each client independently.

Recommendations for this scenario

- Enhance monitoring on metrics that are related to API requests. When you use a managed identity from a gateway, the traceability of the user and client application in Azure OpenAI logs doesn't improve. The gateway should provide logging associated with the request, such as the requesting client and user IDs.
- Ensure that the gateway makes routing decisions to appropriate model deployments based on the client identity when you route multiple client application requests through a gateway to a shared Azure OpenAI instance. For more information, see [Use a gateway in front of multiple Azure OpenAI deployments](#).

Authenticate client applications that access multiple Azure OpenAI instances



Download a [Visio file](#) of this architecture.

Scenario constraints

This scenario has the following constraints:

- Client applications connect to multiple Azure OpenAI instances in one or more regions.
- Clients can't use, or you don't want to use, Microsoft Entra ID or other OIDC providers for authentication.
- You want to use key-based authentication for client applications.

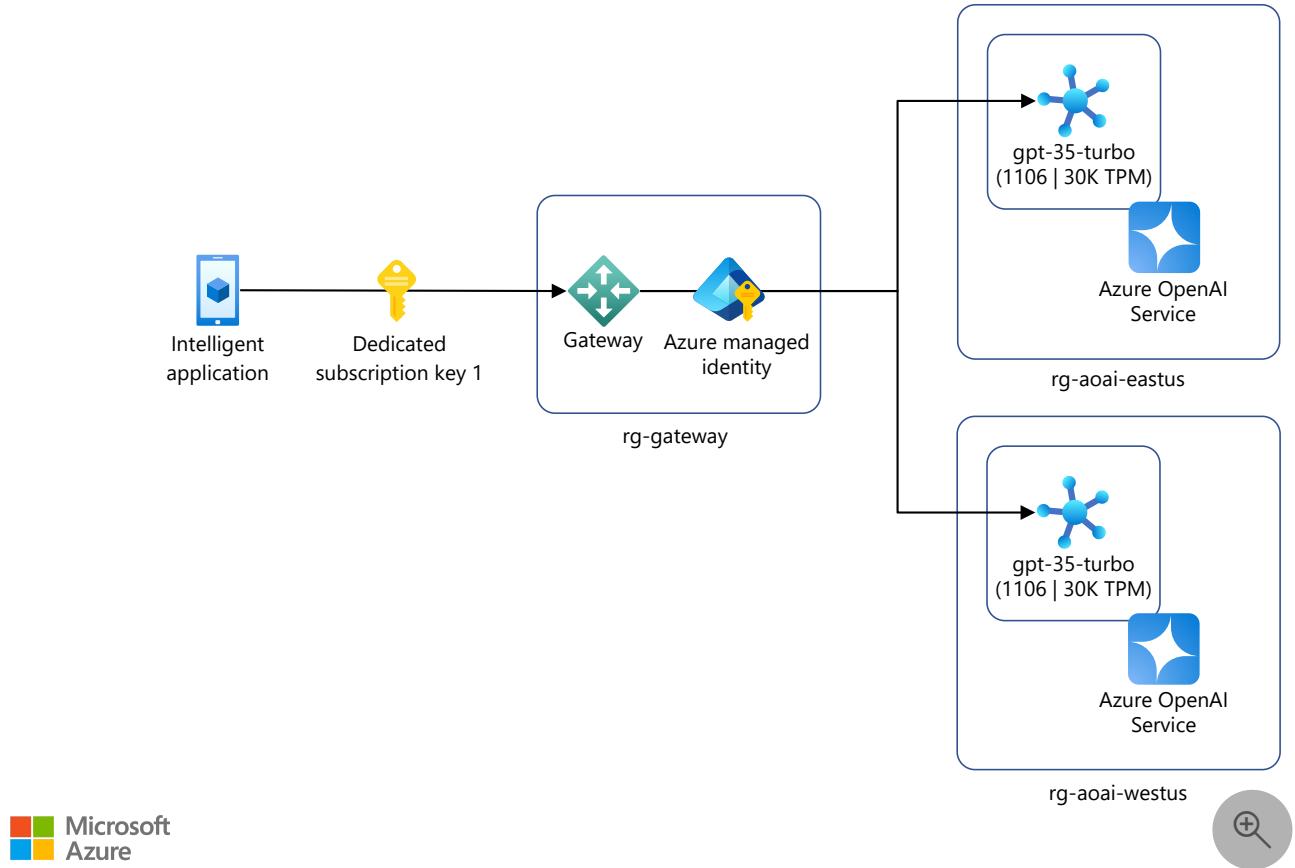
These constraints can apply to the following examples:

- Client applications must distribute their workloads geographically to reduce latency and improve performance.
- Client applications attempt to optimize their tokens per minute quotas by deploying instances across multiple regions.
- An organization requires minimal downtime failover and disaster recovery capabilities to ensure continuous operation. So they manage a dual-deployment strategy, such as a strategy that consists of a provisioned throughput deployment and a pay-as-you-go deployment.
- Client applications must use specific model capabilities that are only available in certain Azure regions.

Connect directly to multiple Azure OpenAI instances

When client applications connect directly to multiple Azure OpenAI instances, each client must store the key for each instance. Along with the security considerations of using keys, there's an increased management burden regarding rotating keys.

Introduce a gateway



Download a [Visio file](#) of this architecture.

When you use a gateway to handle client applications that access multiple Azure OpenAI deployments, you get the same benefits as a gateway that handles [multiple client applications via keys to access a shared Azure OpenAI instance](#). You also streamline the authentication process because you use a single user-defined managed identity to authenticate requests from the gateway to multiple Azure OpenAI instances. Implement this approach to reduce overall operational overhead and minimize the risks of client misconfiguration when you work with multiple instances.

An example of how a gateway is being used in Azure to offload identity to an intermediary is the Azure AI Services resource. In that implementation, you authenticate to the gateway, and the gateway handles authenticating to the differing Azure AI services such as Custom Vision or Speech. While that implementation is similar, it does not address this scenario.

Recommendations for this scenario

- Implement load balancing techniques to distribute the API requests across multiple instances of Azure OpenAI to handle high traffic and ensure high availability. For more information, see [Use a gateway in front of multiple Azure OpenAI deployments or instances](#).

- Correlate token usage for each tenant at the gateway when you implement multitenant scenarios with multiple Azure OpenAI instances. This approach ensures that you track total token usage, regardless of the back-end Azure OpenAI instance that the request is forwarded to.

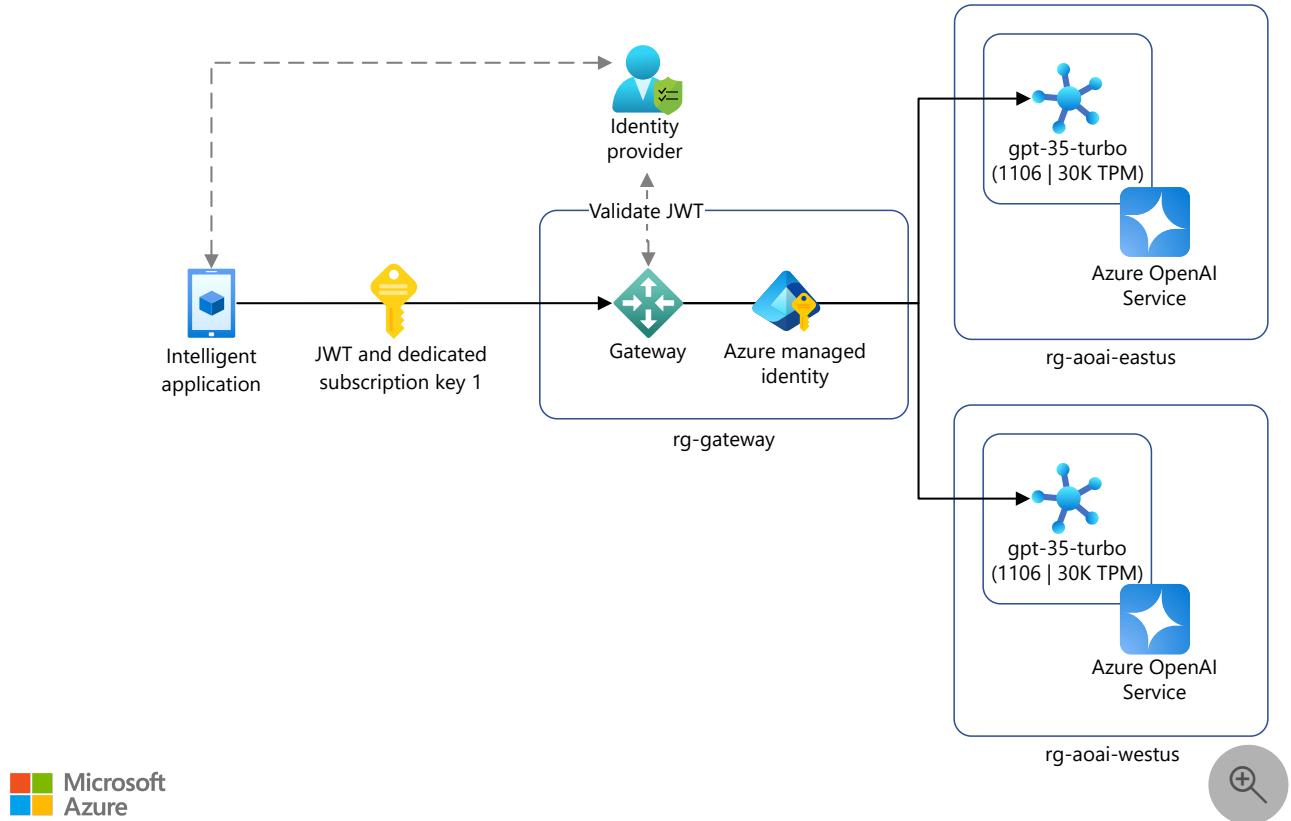
General recommendations

When you integrate Azure OpenAI through a gateway, there are several cross-cutting recommendations to consider that apply in all scenarios.

Use Azure API Management instead of creating your own solution for API orchestration, integration with other Azure services, and cost savings by reducing development and maintenance efforts. API Management ensures secure API management by directly supporting authentication and authorization. It integrates with identity providers, such as Microsoft Entra ID, which enables OAuth 2.0 and provides policy-based authorization. Additionally, API Management uses managed identities for secure and low-maintenance access to Azure OpenAI.

Combine scenarios for a comprehensive gateway solution

In real-world applications, your use cases can span multiple scenarios from this article. For example, you might have client applications that authenticate through an external identity provider and require access to multiple Azure OpenAI instances.



[Download a *Visio file*](#) of this architecture.

To build a gateway that supports your specific requirements, combine the recommendations from these scenarios for a comprehensive approach.

Enforce gateway policies

Before a gateway sends requests to Azure OpenAI instances, make sure you enforce inbound authentication and authorization policies. To ensure that the gateway only forwards authenticated and authorized requests, use user access tokens from an identity provider or certificate validation to implement this approach.

To enable granular control, implement more authorization scoping with roles and permissions for client applications in your gateway. Use these scopes to permit specific operations based on the client application's needs. This approach enhances security and manageability.

For access token validation, be sure to validate all key registered claims such as `iss`, `aud`, `exp`, and `nbf` and any relevant workload-specific claims such as group memberships or application roles.

Use Azure managed identities

To simplify authentication across all client application scenarios, use Azure managed identities to centralize authentication management. This approach reduces the complexity and risks that are associated with managing multiple API keys or credentials in client applications.

Managed identities inherently support Azure RBAC, so they ensure that the gateway only has the lowest level of permissions necessary to access Azure OpenAI instances. To reduce the risk of unauthorized access and simplify compliance with security policies, combine managed identities with other methods that disable alternative authentication.

Implement comprehensive observability

When you implement a gateway with a managed identity, it reduces traceability because the managed identity represents the gateway itself, not the user or the application that makes the request. Therefore, it's essential to improve observability on metrics that are related to API requests. To maintain visibility over access patterns and usage, gateways should provide more tracing metadata, such as the requesting client and user IDs.

Centralized logging of all requests that pass through the gateway helps you maintain an audit trail. A centralized audit trail is especially important for troubleshooting, compliance, and detecting unauthorized access attempts.

Address caching safely

If your API gateway is responsible for caching completions or other inferencing results, make sure the identity of the requestor is considered in the cache logic. Do not return cached results for identities that are not authorized to receive that data.

Gateway implementations

Azure doesn't provide a complete turnkey solution or reference architecture to build the gateway in this article, so you must build and operate the gateway. Azure API management can be used to build a PaaS based solution through built-in and custom policies. Azure also provides examples of community-supported implementations that cover the use cases in this article. Reference these samples when you build your own gateway solution. For more information, see the video [Learn Live: Azure OpenAI application identity and security ↗](#).

Contributors

The following contributors originally wrote this article.

Principal authors:

- [Lizet Pena De Sola](#) | Senior Customer Engineer, FastTrack for Azure
- [Bappaditya Banerjee](#) | Senior Customer Engineer, FastTrack for Azure
- [James Croft](#) | Customer Engineer, ISV & Digital Native Center of Excellence

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Azure RBAC for Azure OpenAI](#)
- [Use managed identities in API Management](#)
- [Policies in API Management](#)
- [Authentication and authorization to APIs in API Management](#)
- [Protect an API in API Management by using OAuth 2.0 and Microsoft Entra ID](#)
- [Secure back-end services by using client certificate authentication in API Management](#)

Related resources

- [Access Azure OpenAI and other language models through a gateway](#)
- [Design a well-architected AI workload](#)

Implement advanced monitoring for Azure OpenAI through a gateway

Azure AI services

Azure OpenAI Service

Azure API Management

Microsoft Entra ID

Azure Monitor

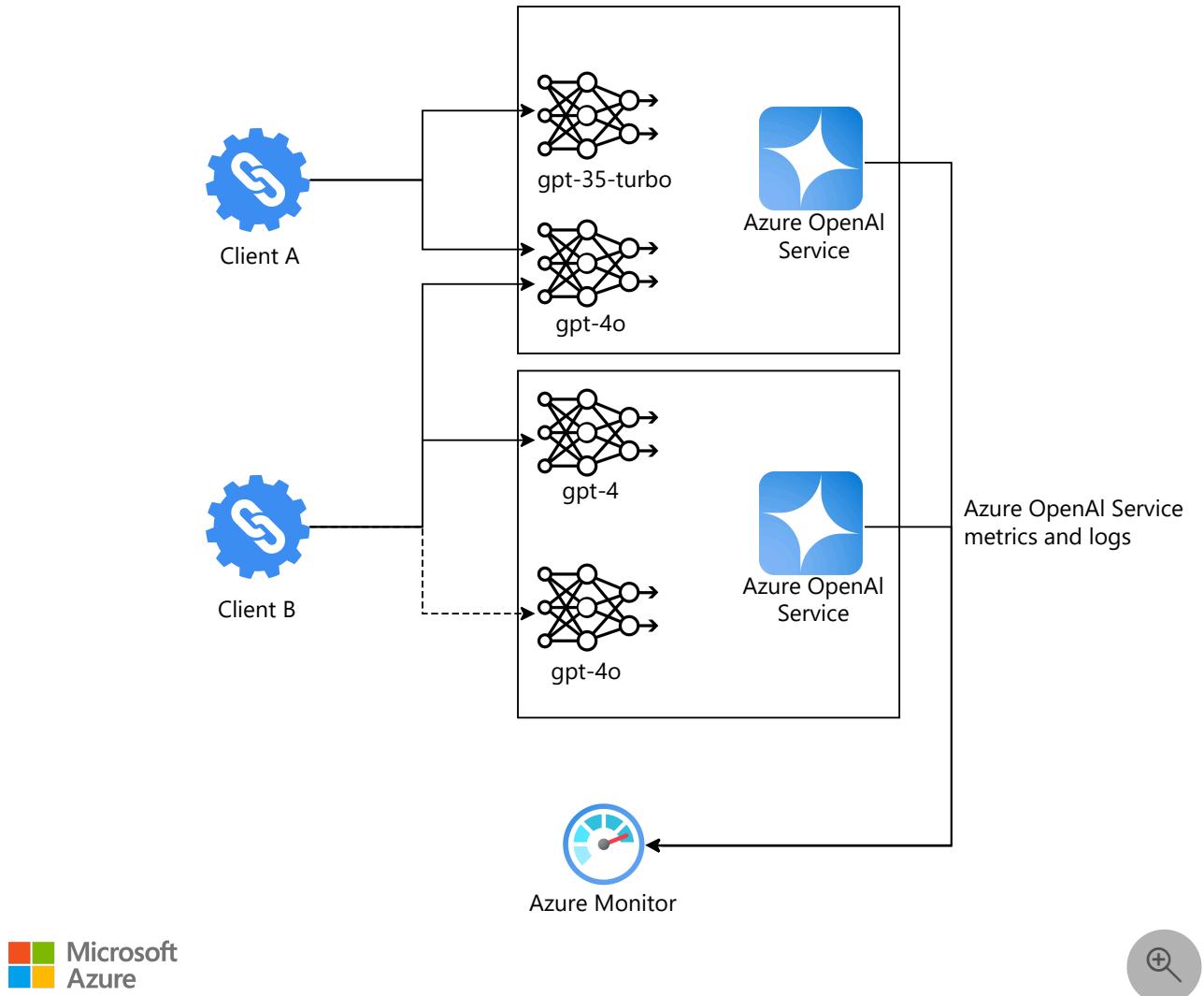
Monitoring workloads that include Azure OpenAI Service can be as simple as enabling diagnostics for Azure OpenAI and using preconfigured dashboards. However, this strategy doesn't satisfy several common, more complex, organizational monitoring requirements for generative AI workloads, such as:

- Tracking usage by client and model to manage quotas and implement chargeback solutions.
- Logging model inputs and model outputs for various auditing use cases and monitoring model performance.
- Performing near real-time monitoring.

 Note

For more information about how to monitor Azure OpenAI directly, see [Monitor Azure OpenAI](#).

The following diagram illustrates the monitoring of Azure OpenAI instances without a gateway. A gateway isn't required for this topology. Your decision to include a gateway depends on whether the outlined monitoring scenarios are part of your requirements. This article examines the challenges that each monitoring scenario addresses, along with the benefits and costs of incorporating a gateway for each scenario.



Track model usage

Many workloads or organizations need to track the usage of Azure OpenAI models by both clients and models across all Azure OpenAI instances. They use this information to:

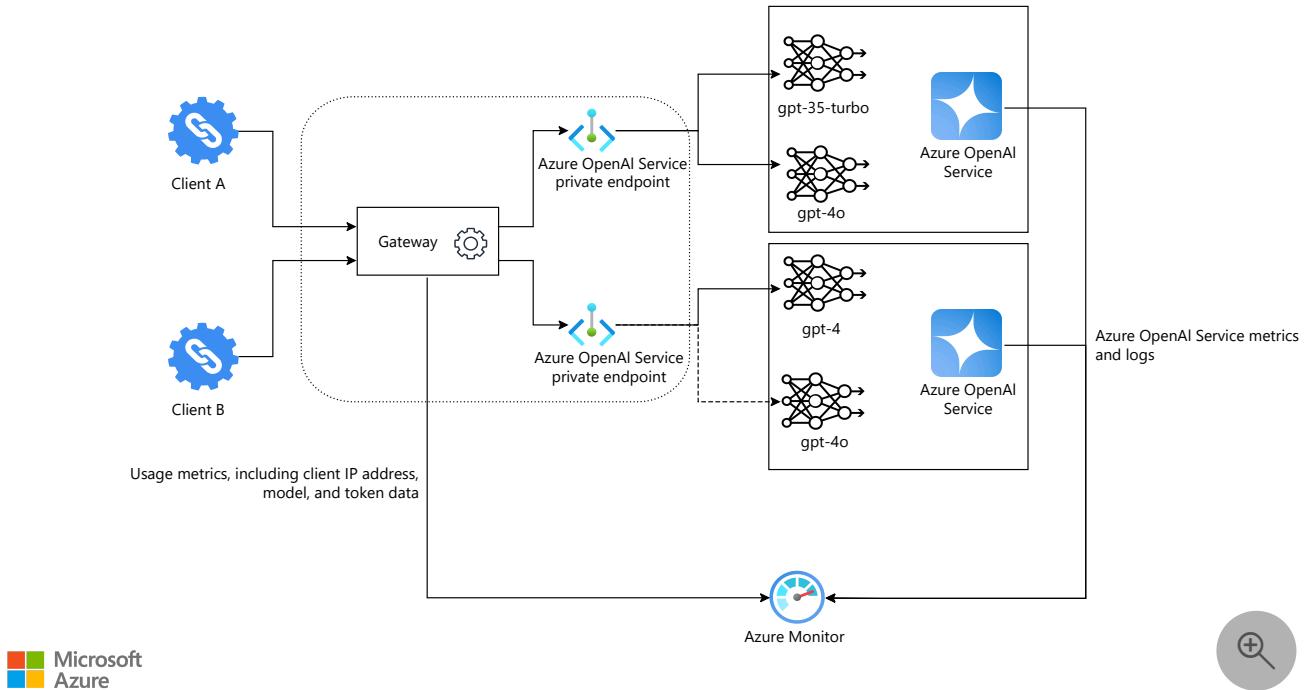
- Implement a chargeback system where they allocate usage costs to the appropriate organization or application owner.
- Budget and forecast for future usage.
- Tie modal cost and usage to model performance.

You can use the native Azure OpenAI monitoring functionality to track telemetry of the service, but there are challenges.

- For chargeback models, you must be able to associate the Azure OpenAI token usage metrics with an application or business unit. Azure OpenAI telemetry includes a calling IP address with the last octet masked. This masking might make establishing this association to an application or business unit challenging.

- Azure OpenAI instances in various regions likely record logs to Azure Monitor instances within their respective local regions. This process requires you to aggregate logs from different Azure Monitor instances to track usage across all Azure OpenAI instances.

Introduce a gateway to track model usage



Introduce a gateway into this topology to capture the full client IP address, the Microsoft Entra ID (or alternative identity) of the client, or a custom identifier for a business unit, a tenant, or an application in one place. You can then use this data to implement a chargeback solution for budgeting and forecasting and to perform cost-benefit analyses of models.

The following examples show usage queries that are possible when you use Azure API Management as a gateway.

Example query for usage monitoring

Kusto

```
ApiManagementGatewayLogs
| where tolower(OperationId) in ('completions_create', 'chatcompletions_create')
| extend modelkey = substring(parse_json(BackendResponseBody)[ 'model' ], 0,
indexof(parse_json(BackendResponseBody)[ 'model' ], '-' , 0, -1, 2))
| extend model = tostring(parse_json(BackendResponseBody)[ 'model' ])
| extend prompttokens = parse_json(parse_json(BackendResponseBody)[ 'usage' ])
[ 'prompt_tokens' ]
| extend completiontokens = parse_json(parse_json(BackendResponseBody)[ 'usage' ])
```

```

['completion_tokens']
| extend totaltokens = parse_json(parse_json(BackendResponseBody)[ 'usage' ])
['total_tokens']
| extend ip = CallerIpAddress
| summarize
    sum(todecimal(prompttokens)),
    sum(todecimal(completiontokens)),
    sum(todecimal(totaltokens)),
    avg(todecimal(totaltokens))
    by ip, model

```

Output:

| Results | Chart | | | |
|-------------------|------------------|--|---|--|
| ip | model | sum_prompttokens | sum_completiontokens | sum_totaltokens |
| > 172.16.0.0 | text-davinci-002 | 9415.0000000000000000000000000000... | 414.0000000000000000000000000000... | 9829.0000000000000000000000000000000000... |
| > 172.31.255.255 | text-davinci-002 | 69.00000000000000000000000000000000... | 36.00000000000000000000000000000000... | 105.00000000000000000000000000000000000000... |
| > 172.31.255.255 | code-cushman-001 | 81.00000000000000000000000000000000... | 599.00000000000000000000000000000000... | 680.00000000000000000000000000000000000000... |
| > 172.31.255.255 | text-curie-001 | 61.00000000000000000000000000000000... | 35.00000000000000000000000000000000... | 96.00000000000000000000000000000000000000... |
| > 192.168.255.255 | text-davinci-002 | 5689.00000000000000000000000000000000... | 274.00000000000000000000000000000000... | 5963.00000000000000000000000000000000000000... |



Example query for prompt usage monitoring

Kusto

```

ApiManagementGatewayLogs
| where tolower(OperationId) in ('completions_create', 'chatcompletions_create')
| extend model = tostring(parse_json(BackendResponseBody)[ 'model' ])
| extend prompttokens = parse_json(parse_json(BackendResponseBody)[ 'usage' ])
['prompt_tokens']
| extend prompttext = substring(parse_json(parse_json(BackendResponseBody)
['choices']))[0], 0, 100)

```

Output:

| Results | Chart | | |
|-----------------------------|------------------|--------------|--|
| TimeGenerated [UTC] | model | prompttokens | prompttext |
| > 1/27/2023, 5:16:33.683 PM | text-davinci-002 | 39 | [{"text": "\n1. The Maldives\n2. Robinson Crusoe Island, Chile\n3. North Sentinel Island, India\n4. ..."}] |
| > 1/23/2023, 9:19:16.896 PM | text-davinci-002 | 39 | [{"text": "\n1. Maldives\n2. Robinson Crusoe Island\n3. Easter Island\n4. The Galapagos Islands\n..."}] |
| > 1/23/2023, 9:19:21.802 PM | text-davinci-002 | 23 | [{"text": "The price for bubblegum did not increase on Thursday.", "index": 0, "finish_reason": "stop", "i..."}] |
| > 1/23/2023, 9:19:22.508 PM | text-davinci-002 | 39 | [{"text": "\n1. Maldives\n2. Seychelles\n3. Bora Bora\n4. Moorea\n5. Tahiti\n6. Fiji\n7. Palawan\n8. B..."}] |
| > 1/23/2023, 9:19:29.911 PM | text-davinci-002 | 1849 | [{"text": "\n\nMMM closed at 84.72000122070312 on 2010-01-20.", "index": 0, "finish_reason": "stop", "..."}] |
| > 1/23/2023, 9:19:33.251 PM | text-davinci-002 | 1850 | [{"text": "\n\nThe average adj price of MMM in January 2010 was 59.826175689697266.", "index": 0, "..."}] |
| > 1/23/2023, 9:22:26.817 PM | text-davinci-002 | 1850 | [{"text": "\n\nThe average adj price of MMM in January 2010 was 59.47.", "index": 0, "finish_reason": "..."}] |



Audit model inputs and outputs

Central to many auditing requirements for generative AI workloads is monitoring the input and output of the models. You might need to know whether a response was from a model or sourced from a cache. There are multiple use cases for monitoring both inputs and outputs of models. In most scenarios, auditing rules should be applied uniformly across all models for both inputs and outputs.

The following use cases are for monitoring the inputs to models.

- **Threat detection:** Analyze inputs to identify and mitigate potential security risks.
- **Usage guidelines violation detection:** Analyze inputs for offensive language or other usage standards to ensure that the system is professional, safe, and unbiased.
- **Model performance:** Combine with model outputs to evaluate performance on metrics like groundedness and relevance. You can use this information to address performance problems with the model or prompts.

The following are some of the use cases for monitoring the outputs to models.

- **Data exfiltration detection:** Analyze outputs to guard against unauthorized transfer of sensitive information.
- **Stateful compliance:** Monitor outputs over multiple interactions within the same conversation to detect stealthy leaks of sensitive information.
- **Compliance:** Ensure that outputs adhere to corporate guidelines and regulatory requirements. Two examples are that models don't provide legal advice or make financial promises.
- **Model performance:** Combine with model inputs to evaluate performance on metrics like groundedness and relevance. You can use this information to address performance problems with the model or prompts.

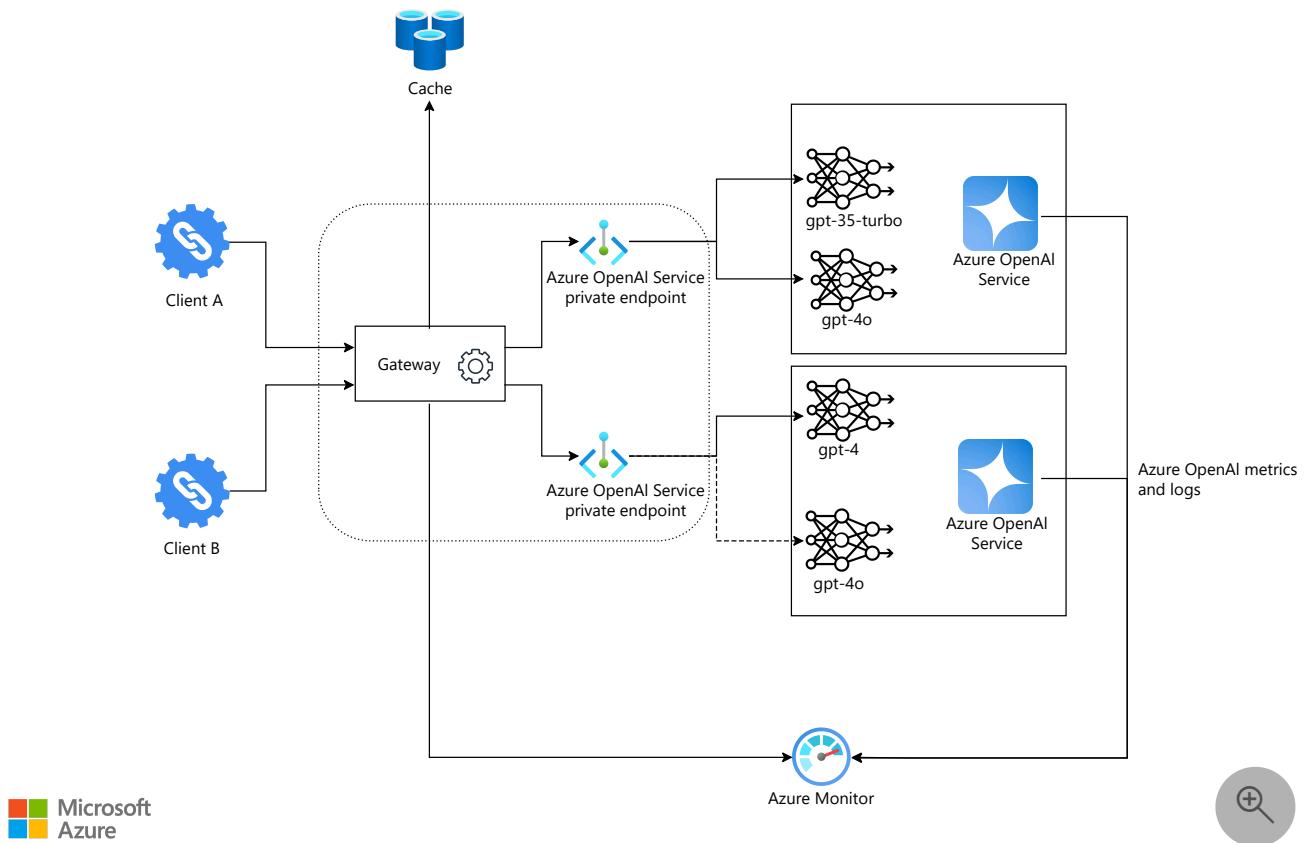
Challenges to auditing model inputs and outputs directly from the model

- **Model logging constraints:** Some services such as Azure OpenAI don't log model inputs and outputs.
- **Cache:** More complex architectures might serve responses from cache. In those scenarios, the model isn't called and doesn't log the input or output.
- **Stateful conversations:** The state of a multiple-interaction conversation might be stored outside the model. The model doesn't know which interactions should be correlated as a

conversation.

- **Multiple-model architecture:** The orchestration layer might dynamically invoke multiple models to generate a final response.

Introduce a gateway for auditing model inputs and outputs



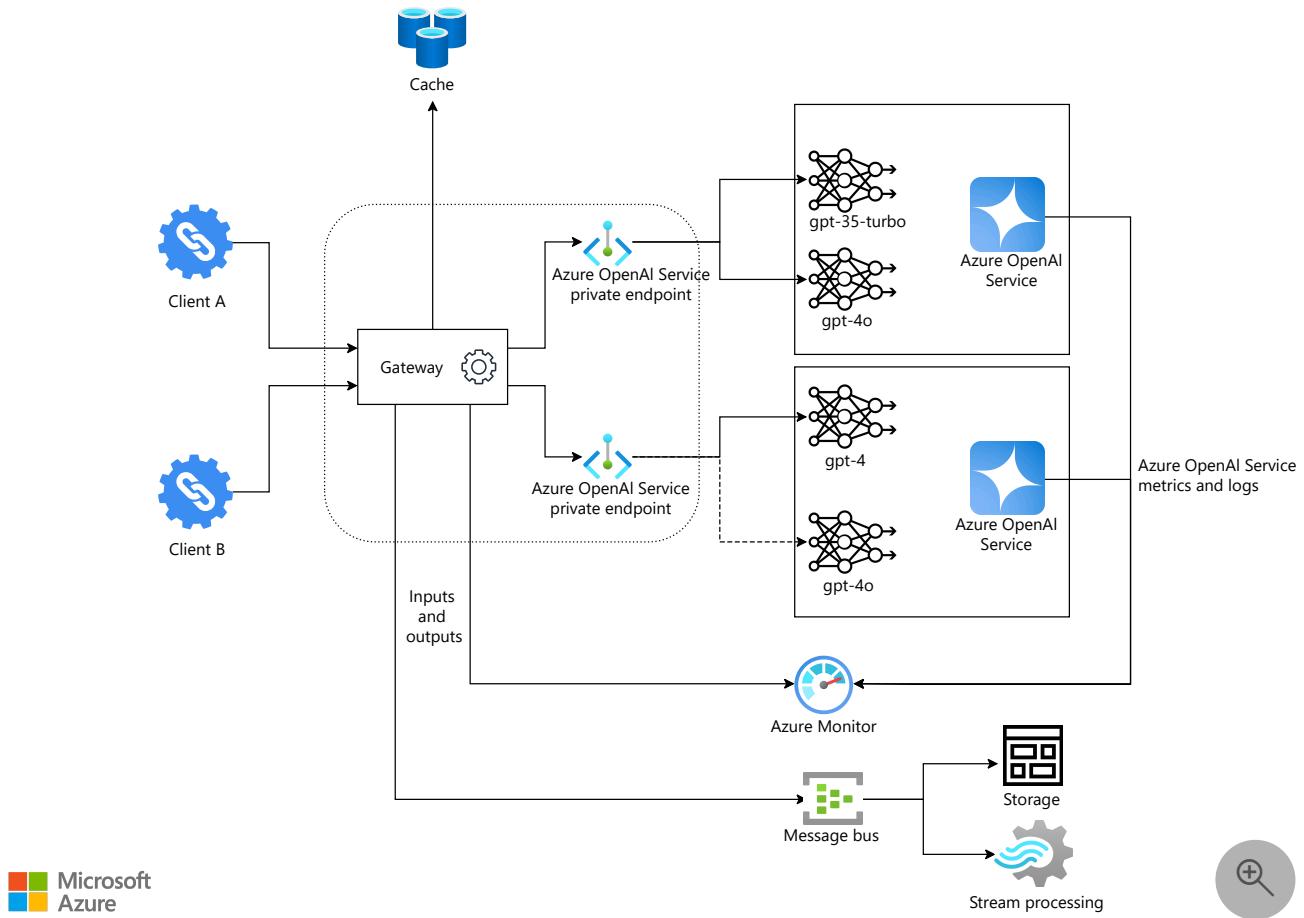
Introduce a gateway into this topology to capture both the original input directly from the client and the final output returning to the client. Because the gateway is an abstraction between the client and the models and directly receives the request from the clients, the gateway is in a position to log that raw, unprocessed request. Likewise, because the gateway is the resource that returns the final response to the client, it's also able to log that response.

The gateway has the unique capability to log both the client's request and the final response that it received. This feature applies whether the response came directly from a model, was aggregated from multiple models, or was retrieved from cache. Further, if the clients pass a conversation identifier, the gateway can log that identifier with the input and output. You can use this implementation to correlate multiple interactions of a conversation.

Monitoring inputs and outputs at the gateway allows you to apply auditing rules uniformly across all models.

Near real-time monitoring

Azure Monitor isn't optimized for near real-time processing because of the inherent [latency in log data ingestion](#). If your solution requires near real-time processing of traffic, you can consider a design where you publish logs directly to a message bus and use a stream processing technology, such as Azure Stream Analytics, to perform windowed operations.



Considerations when introducing a gateway for monitoring

- **Latency:** Introducing a gateway into your architecture adds latency to your responses. You need to ensure that the observability benefits outweigh the performance implications.
- **Security and privacy:** Ensure that the monitoring data gathered by using the gateway continues to adhere to customer privacy expectations. Observability data must adhere to the established security and privacy expectations of the workload and not violate any customer privacy standards. Continue to treat any sensitive data captured through monitoring as sensitive data.
- **Reliability:** Determine whether the monitoring function is crucial to the functionality of the workload. If it is, the entire application should be down when the monitoring system is unavailable. If it isn't crucial, the application should continue to work in an unmonitored state if the monitoring system is down. Understand the risks of adding a new single point

of failure, either through service faults or human-caused configuration problems in the gateway.

- **Implementation:** Your implementation can take advantage of out-of-the-box gateways like API Management, including all the required configurations. Another common approach is to implement an orchestration layer through code.

Reasons to avoid introducing a gateway for monitoring

If a single application accesses a single model, the added complexity of introducing a gateway likely outweighs the benefits of monitoring. The client can handle the responsibility of logging inputs and outputs. And you can take advantage of native logging capabilities of the model or service that you use. The gateway becomes beneficial when you have multiple clients or multiple models that you need to monitor.

Next steps

A gateway implementation for your workload provides benefits beyond the tactical, multiple back-end routing benefit described in this article. For more information, see [Access Azure OpenAI and other language models through a gateway](#).

Related resources

- [Design a well-architected AI workload](#)
- [API gateway in API Management](#)

Design to support foundation model life cycles

05/17/2025

Foundation models are versioned dependencies that you use in your AI workload. Each foundation model has a life cycle that must be considered. Like code libraries and other dependencies in your workload, foundation models receive minor version updates that provide performance enhancements and optimizations. Major version updates introduce substantive changes to capabilities, performance, or training data freshness. Over time, foundation models might be deprecated because of obsolescence or your model's host preferences that are beyond your control.

You must design your workload to support the documented life cycles of the models that you choose as dependencies. If you don't consider the models' life cycles, you might unnecessarily perform risky upgrades, introduce untested behavior changes, take unnecessary workload downtime, or experience outages because of the way that your hosting platform handles end-of-life models.

A workload that's designed to support the life cycles of its models makes it easier to experiment and safely migrate to new foundation models as they enter the marketplace.

Types of model updates

The scope of the model update in your generative AI solution can vary drastically, from upgrading for a minor revision change to choosing a new model family. There are various reasons why you might choose to upgrade the model in your solution. The following table lists different update scopes, along with an example and benefits of making this upgrade.

 Expand table

| Scope of change | Benefits of updating the model | Example |
|-----------------------------|--|--|
| Minor version update | Delivers improved performance and refined capabilities, usually without requiring significant changes to your existing implementation | Moving from GPT-4o v2024-08-06 to GPT-4o v2024-11-20 |
| Intermediate version update | Provides substantial performance improvements, new capabilities, and enhanced reliability while maintaining most backward compatibility and requiring only moderate implementation adjustments | Moving from GPT-3 to GPT-3.5 |

| Scope of change | Benefits of updating the model | Example |
|-----------------------------|---|--|
| Major version update | Delivers transformational improvements in reasoning, capabilities, context size, and performance that justify the significant implementation changes and adjustment of prompts | Moving from GPT-3 to GPT-4 |
| Variant update | Provides specialized optimizations, such as increased processing speed and reduced latency, while maintaining the core architecture and usually ensuring backward compatibility with the base model | Moving from GPT-4 to GPT-4-Turbo |
| Generational version update | Delivers significant improvements in reasoning, multimodal capabilities, and performance that fundamentally expand the model's utility while potentially requiring complete reimaging of implementation strategies | Moving from GPT-4 to GPT-4o |
| General model change | Provides access to specialized capabilities, different price-performance ratios, and potentially better alignment with specific use cases | Moving from GPT-4 to DeepSeek |
| Specialized model change | Provides domain-specific optimization, enhanced performance for particular tasks, and potentially lower costs compared to using general-purpose models for specialized applications | Moving from GPT-4 to Prizedata |
| Deployment option change | Provides greater control over infrastructure, customization options, and potential cost savings while allowing for specialized optimization and enhanced data privacy at the expense of increased management responsibility | Moving from LLaMa-1 hosted as a managed online endpoint in Azure AI Foundry to self-hosting LLaMa-1 on a virtual machine |

As illustrated in the table, the benefits of moving to a new model are typically a combination of the following factors:

- Performance, such as speed and latency
- Capacity, such as throughput that's measured in transactions per minute
- Quota availability
- Cost efficiency
- Regional availability
- Multimodal capabilities
- Updated training knowledge

- Bug fixes
- Specialization or despecialization to better align with your use case
- Avoiding workload outages because of model host life cycle policies

Model retirement behavior

Model retirement behavior depends on your model deployment strategy. There are three key strategies for deploying models. It's important to understand how each strategy handles version retirements:

- **MaaS (model as a service) solutions** are pretrained models exposed as APIs that provide scalability and ease of integration. They have a trade-off of potentially higher costs and lower control of models. Examples of MaaS solutions include models deployed in Azure OpenAI Service and models from the model catalog deployed as serverless APIs.
- **MaaP (model as a platform) solutions** are models deployed and managed within a larger platform, such as models from the Azure model catalog deployed in [managed compute](#). This solution usually provides greater control of models but requires more management than MaaS solutions.
- **Self-hosting models** are models deployed on your own infrastructure. This deployment provides maximum control over models but requires significant responsibility for infrastructure, management, and maintenance.

Both MaaS and MaaP strategies in Azure source models from the Azure AI model catalog. Models in the model catalog follow a life cycle where models are [deprecated](#) and then [retired](#). You must plan for these eventualities in your workload.

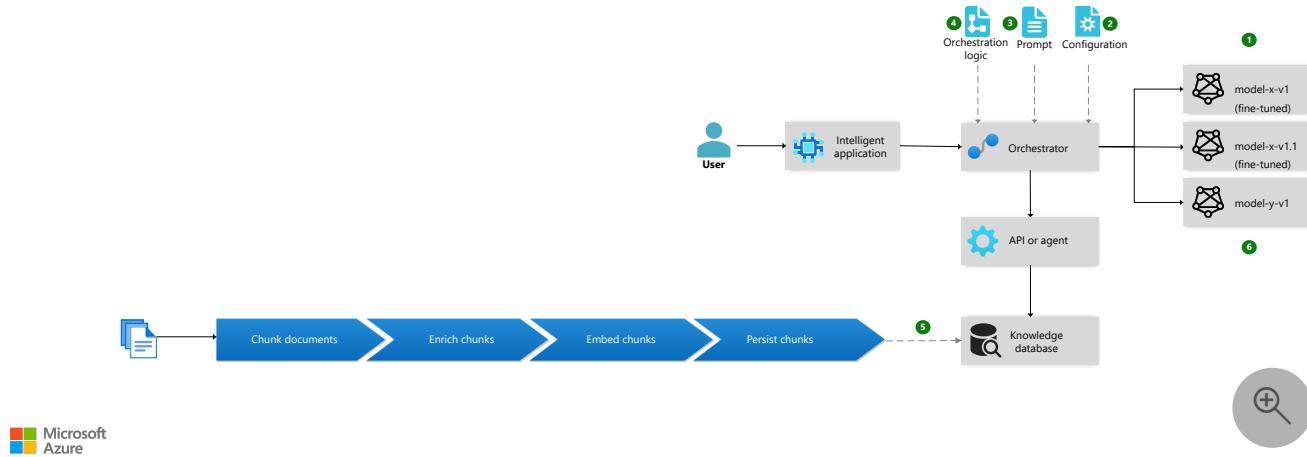
Warning

For MaaS services, including Azure OpenAI-deployed models and models deployed by using the serverless API model, it's crucial to understand that existing deployments for *retired* models return HTTP errors. If you don't upgrade to a supported model, your application no longer operates as expected. For *deprecated* models, you can't create new deployments for those models, but existing deployments continue to work until they're retired. For more information, see [Serverless API model deprecations and retirements](#) and [Azure OpenAI model deprecations and retirements](#).

When you self-host models or use managed compute, you maintain full control and aren't required to update models. But you might want to replicate a model life cycle for the added

benefits that a newer model can bring to your workload.

Breadth of change for model updates



You need to evaluate how a model update affects your workload so that you can plan the transition from the old model to the new model. The extent of workload change depends on the functional and nonfunctional differences between the old and new models. The diagram shows a simplified chat architecture that has numbered sections that highlight areas where a model update might have an effect.

For each of these areas, consider downtime caused by updates and how you handle any requests that the system is processing. If maintenance windows are available for your workload, use those windows when the scope of change is large. If maintenance windows aren't available, address changes in these areas to maintain your workload's functionality and service-level objectives during the model update.

1. **The model:** The obvious change is to the model itself. You deploy the new model by using your chosen model deployment strategy. You need to evaluate trade-offs between in-place upgrades versus side-by-side deployment.

When you move to a new model revision from a fine-tuned model, you need to perform the fine-tuning on the new model version again before you use it. When you update to use a different model, you need to determine if fine-tuning is required.

2. **The model configuration:** When you update the foundation model in your AI solution, you might need to adjust hyperparameters or configurations to optimize performance for the new model's architecture and capabilities. For example, switching from a transformer-based model to a recurrent neural network might require different learning rates and batch sizes to achieve optimal results.
3. **The prompt:** When you change foundation models in your workload, you might need to adjust system prompts in your orchestrators or agents to align with the new model's

strengths and capabilities.

Along with updating the model configuration, updating the prompt is one of the most common changes when you update models. When you evaluate a new model, even for a minor version update, test changes to the prompt if it doesn't meet your requirements. This approach ensures that you address performance problems before exploring other modifications. You need to address the prompt when you change to new models. It's also likely that you need to address the prompt when you make large revision changes.

4. **The orchestration logic:** Some model updates, especially when you take advantage of new features, require you to make changes to your orchestration or agent logic.

For example, if you update your model to GPT-4 to take advantage of [function calling](#), you have to change your orchestration logic. Your old model returned a result that you could return to the caller. With function calling, the call to the model returns a function that your orchestration logic must call. In Azure, it's typical to implement orchestration logic in [Azure AI Agent Service](#) or by using code-based solutions like [Semantic Kernel](#) or [LangChain](#) hosted in Azure.

5. **The grounding data:** Some model updates and larger scoped changes might require you to make changes to your grounding or fine-tuning data or how you retrieve that data.

For example, when you move from a generalized model to a domain-specific model, such as a model focused on finance or medicine, you might no longer need to pass domain-specific grounding data to the model. Another example is when a new model can handle a larger context window. In this scenario, you might want to retrieve other relevant chunks or tune the size of your chunks. For more information, see [Design and develop a retrieval-augmented generation \(RAG\) solution](#).

6. **Hardware:** For models that run in MaaP, a model change might require new hardware. Only specific compute SKUs are enabled for models from the catalog. Also, hardware can be deprecated, which requires you to run the model on new hardware.

Design for change

You'll most likely update models in your workload. If you use the MaaS deployment strategy in Azure, models are retired and you need to upgrade to a newer version. You might also choose to move to different models or model versions to take advantage of new features, lower pricing, or improved performance. Either way, your architecture must support updating the model that your generative AI workload uses.

The previous section discussed the [different breadths of change](#). You should follow proper machine learning operations (MLOps), data operations (DataOps), and generative AI operations

(GenAIOps) practices to build and maintain automated pipelines for model fine-tuning, engineer prompts, change hyperparameters, and manage orchestration logic.

Updates to the hyperparameters and prompts are common in most model updates. Because these changes are so common, your architecture should support a controlled change mechanism for these areas. An important consideration is that prompts and hyperparameters are designed for specific model versions. You need to ensure that the prompts and hyperparameters are always used with the correct model and version.

Automated pipelines

Implement automated pipelines to test and evaluate the differing aspects of your generative AI application:

- **MLOps:** Follow the [Azure MLOps guidance](#) to build pipelines for model fine-tuning, if applicable.
- **GenAIOps:** Implement GenAIOps pipelines to [test and evaluate changes to the model, model hyperparameters, prompt, and orchestration logic](#).
- **DataOps:** Implement [DataOps](#) pipelines to [test and evaluate changes to your RAG grounding data](#).

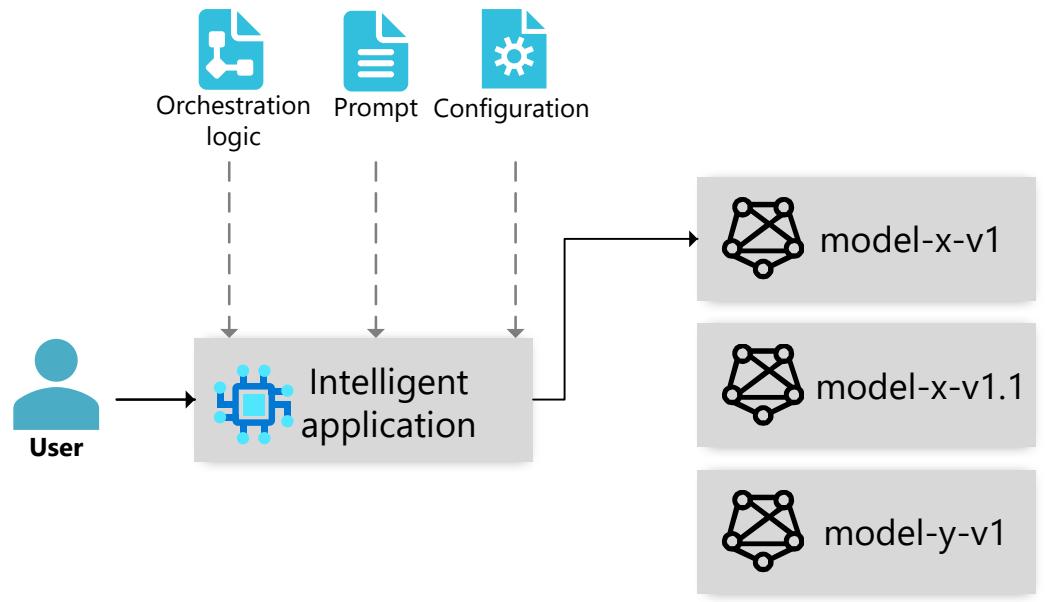
You should implement pipelines for the following reasons:

- To help you in your iterative development and experimentation (inner loop)
- To perform safe deployment and operationalization of your generative AI solution (outer loop)

When possible, use the same baseline data that you use with the production application to test the changes to your generative AI application. This approach might not be possible if the updated application uses new model features that require a change to the data.

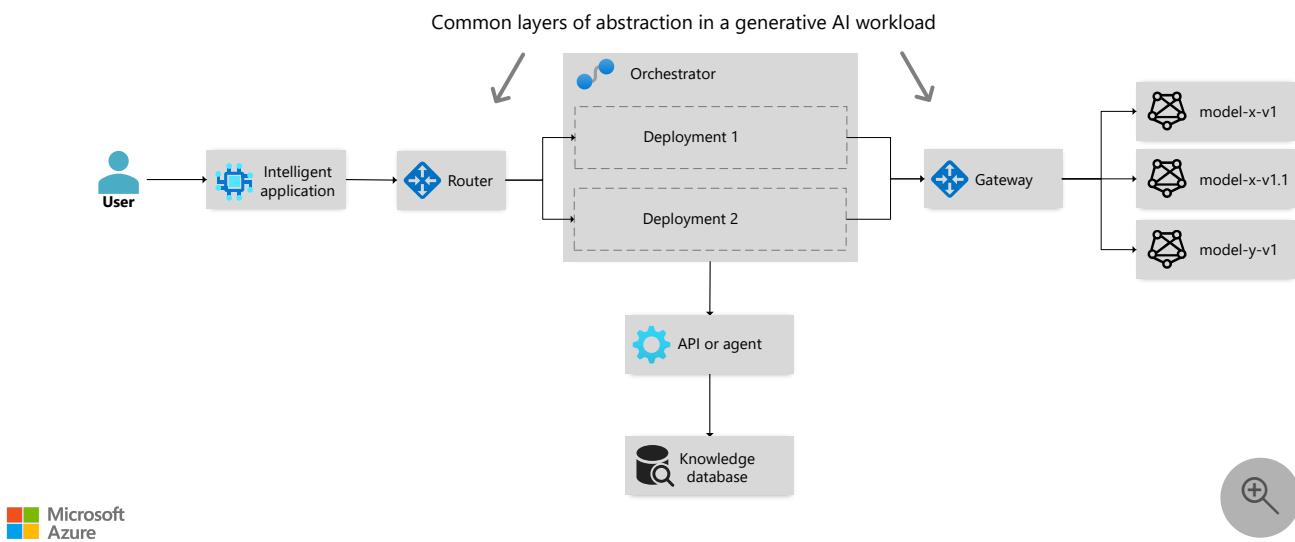
Architecture considerations

In simple architectures, like the following architecture, the client directly calls the model with the correct prompt version and configuration. If there are changes to the prompt, a new client is deployed with the new prompt, and it calls the new model. Linking the prompt, configuration, and model versions isn't a challenge.



Production architectures aren't simple. You generally implement an orchestrator or agent whose responsibility is to manage the flow of the interactions between any knowledge database and the models. Your architecture might also implement one or more layers of abstraction, such as a router or a gateway:

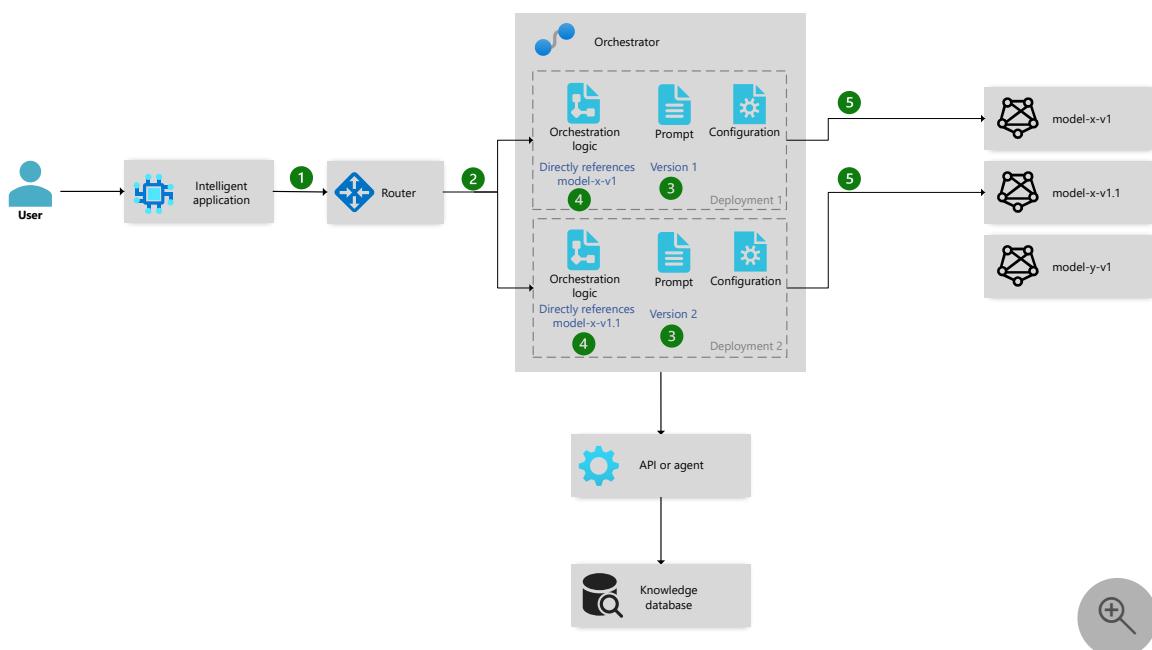
- **Router:** A router directs traffic to different orchestrator deployments. A router is useful in deployment strategies such as blue-green deployments where you might choose to route a specific percentage of traffic to a new orchestrator version as part of your safe deployment practices. This component can also be used for A/B testing or traffic mirroring to evaluate tested, but unvalidated, changes with production traffic.
- **Gateway:** It's common to [proxy access to AI models for various reasons](#). For example, you might load balance or enable failover between multiple back-end instances, implement custom authentication and authorization, or implement logging and monitoring for your models.



Because of the layers of indirection involved, your architecture must be designed to support sending specific versions of prompts to specific models or model versions. For instance, you might have a prompt in production, such as prompt1, that's designed for a model, such as model1. If you upgrade to model1.1, you might need to update prompt1 to prompt2. In this example, your architecture needs to always use prompt1 with model1 and prompt2 with model1.1.

Router

The following diagram illustrates an architecture that uses a router to route requests to multiple deployments. Another [example of this architecture includes Azure AI Foundry](#) and uses a managed online endpoint as the router. And the different versions of the orchestrator are deployed to managed compute.

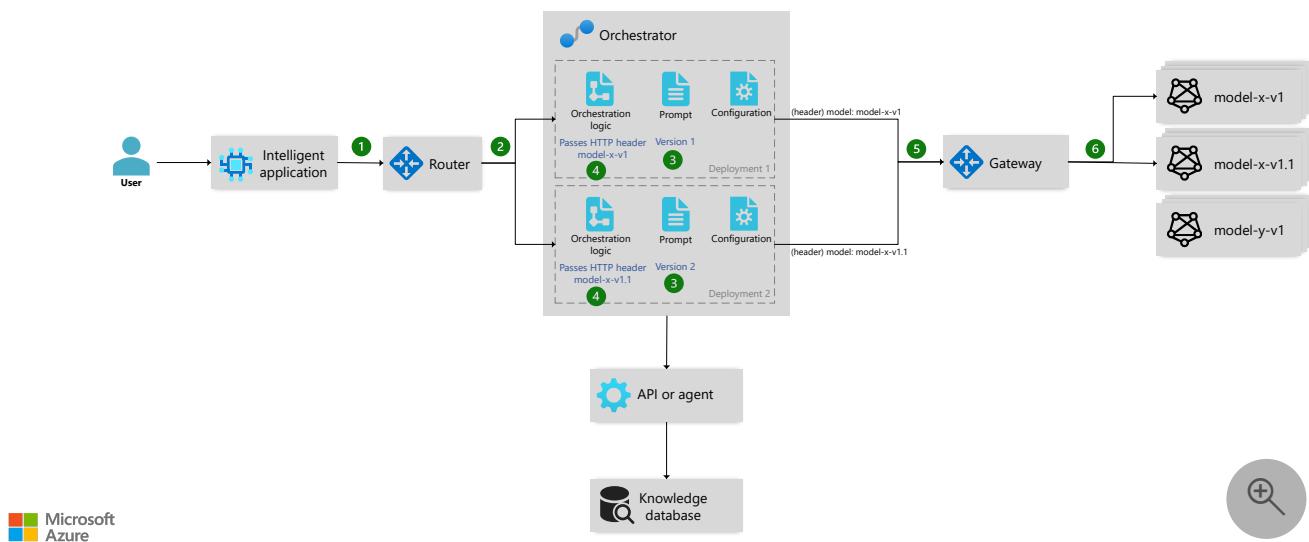


The following flow describes how different deployments of an orchestrator call the correct model. Each deployment has its own version of model configuration and a prompt:

1. A user issues a request from an intelligent application and that request is sent to a router.
2. The router routes to either Deployment 1 or Deployment 2 of the orchestrator, depending on its logic.
3. Each deployment has its own version of the prompt and configuration.
4. The orchestrator is configured with the specific model and version. It uses this information to call the appropriate model and version directly.
5. Because the specific version of the prompt is deployed along with the orchestrator that's configured with the specific model and version, the correct prompt is sent to the correct model version.

Gateway

The following diagram illustrates an architecture that uses a router to route requests to multiple deployments. However, in this architecture, all model requests are routed through a gateway. It's common to [proxy access to AI models for various reasons](#), including load balancing, enabling failover between multiple back-end instances in a single region or multiple regions, and implementing a provisioned throughput unit with a pay-as-you-go spillover strategy.



The following flow describes how different deployments of an orchestrator call the correct model through a gateway. Each deployment has its own version of model configuration and a prompt:

1. A user issues a request from an intelligent application and that request is sent to a router.

2. The router routes to either Deployment 1 or Deployment 2, depending on its logic.
3. Each deployment has its own version of the prompt.
4. The orchestrator is configured with the specific model and version. It uses this information to set an HTTP header that indicates the correct model and version to call.
5. The orchestrator calls the gateway. The request contains the HTTP header that indicates the name and version of the model to use.
6. The gateway uses the HTTP header to route the request to the appropriate model and version. It might also apply configuration defined at the gateway.

Externalize configuration

The [External Configuration Store](#) cloud design pattern is a good way to handle storing prompts and configuration. For some scopes of model changes, you might be able to coordinate the model deployment with the system prompt and configuration changes if they're stored in an updatable location outside of your orchestrator or agent's code. This approach doesn't work if you have orchestration logic to adjust, but is useful in many smaller scope model updates.

Compute choice

For MaaP hosting, models are often limited to a subset of host-provided compute resources. All compute is subject to quotas, availability constraints, and end-of-life announcements. Use the routing patterns to support transition to new hardware when your current hardware is no longer supported or there are constraints that prevent extra scale-out.

An example of an end-of-life announcement is the [NC A100 v4 series of compute announcement](#). If you host models on this hardware, you have to transition to another supported SKU that isn't end-of-life and has more availability. This transition might also concurrently require a model change if the new SKU doesn't support your current model.

Recommendations

- Add layers of abstraction and indirection to enable controlled modifications to specific areas of your workload. These areas include the client, intelligent application API, orchestration, model hosting, and grounding knowledge.
- All changes to model versions, prompts, configurations, orchestration logic, and grounding knowledge retrieval must be tested before use in production. Ensure that tested combinations are *pinned together* in production, which means that they remain

tightly linked when deployed. A/B testing, load balancing, and blue-green deployments must not mix components to avoid exposing users to untested combinations.

- Test and evaluate new versions and new models by using automated pipelines. You should compare the results to the results of your baseline to ensure that you get the results that you require.
- Be intentional about updating models. Avoid using platform features that automatically upgrade production models to new versions without the opportunity to test. You need to be aware of how every model update affects your workload. If you use [Azure AI model inference](#), set your deployments with a specific version and don't provide an upgrade policy. This setup requires a manual upgrade if a new version is released. For Azure OpenAI, set deployments to [No Auto Upgrade](#) to turn off automatic upgrades.
- Ensure that your observability and logging solution collects enough metadata to correlate observed behavior with the specific prompt, configuration, model, and data retrieval solution involved. This correlation enables you to identify unexpected regressions in system performance.

Summary

There are various reasons to update the foundational model in your generative workload. These reasons range from required version upgrades when models are retired to the decision to switch to a different model. Depending on the scope of the model update, you might need to implement and evaluate changes to the model, including changes to the prompt, model configuration, orchestration logic, or data pipeline. You should follow MLOps, DataOps, and GenAIOps guidance to build automated workflows for the different aspects of your solution. Automated workflows enable you to test, evaluate, and deploy new versions. You also need to ensure that your architecture supports running multiple versions of an orchestrator where each version links its configuration and prompt version to a specific model version.

Your architecture should support updates to new or different models and any necessary changes to the prompt or model configuration, without requiring modifications to the intelligent application or the user experience. These updates should be encapsulated within their appropriate components and your operations should automate the testing, evaluation, and deployment of those changes.

Contributors

Microsoft maintains this article. The following contributors wrote this article.

- [Ritesh Modi](#) | Principal Software Engineer

To see nonpublic LinkedIn profiles, sign in to LinkedIn.

Next step

- [Azure OpenAI model deprecations and retirements](#)

Related resources

- [Baseline OpenAI end-to-end chat reference architecture](#)
- [MLOps](#)
- [GenAIOps for MLOps practitioners](#)