

# Baseline OpenAI end-to-end chat reference architecture

[Azure OpenAI Service](#)   [Azure Machine Learning](#)   [Azure App Service](#)   [Azure Key Vault](#)   [Azure Monitor](#)

Enterprise chat applications can empower employees through conversational interaction. This is especially true due to the continuous advancement of language models, such as OpenAI's GPT models and Meta's LLaMA models. These chat applications consist of a chat user interface (UI), data repositories that contain domain-specific information pertinent to the user's queries, language models that reason over the domain-specific data to produce a relevant response, and an orchestrator that oversees the interaction between these components.

This article provides a baseline architecture for building and deploying enterprise chat applications that use [Azure OpenAI Service language models](#). The architecture employs Azure Machine Learning prompt flow to create executable flows. These executable flows orchestrate the workflow from incoming prompts out to data stores to fetch grounding data for the language models, along with other required Python logic. The executable flow is deployed to a Machine Learning compute cluster behind a managed online endpoint.

The hosting of the custom chat user interface (UI) follows the [baseline app services web application](#) guidance for deploying a secure, zone-redundant, and highly available web application on Azure App Services. In that architecture, App Service communicates to the Azure platform as a service (PaaS) solution through virtual network integration over private endpoints. The chat UI App Service communicates with the managed online endpoint for the flow over a private endpoint. Public access to the Machine Learning workspace is disabled.

## Important

The article doesn't discuss the components or architecture decisions from the

[baseline App Service web application](#). Read that article for architectural guidance on how to host the chat UI.

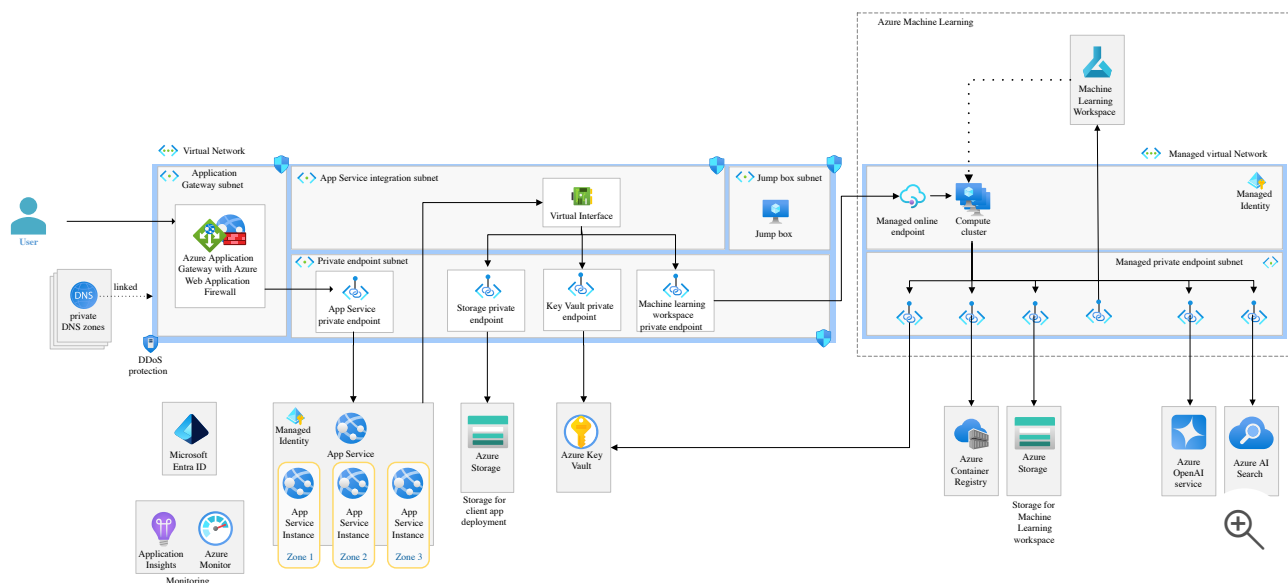
The Machine Learning workspace is configured with [managed virtual network isolation](#) that requires all outbound connections to be approved. With this configuration, a managed virtual network is created, along with managed private endpoints that enable connectivity to private resources, such as the workplace Azure Storage, Azure Container Registry, and Azure OpenAI. These private connections are used during flow authoring and testing, and by flows that are deployed to Machine Learning compute.

### Tip



This article is backed by a [reference implementation](#) which showcases a baseline end-to-end chat implementation on Azure. You can use this implementation as a basis for custom solution development in your first step toward production.

## Architecture



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

# Components

Many of the components of this architecture are the same as the resources in the [baseline App Service web application architecture](#) because the method that you use to host the chat UI is the same in both architectures. The components highlighted in this section focus on the components used to build and orchestrate chat flows, data services, and the services that expose the language models.

- [Machine Learning](#) is a managed cloud service that you can use to train, deploy, and manage machine learning models. This architecture uses several other features of Machine Learning that are used to develop and deploy executable flows for AI applications that are powered by language models:
  - [Machine Learning prompt flow](#) is a development tool that you can use to build, evaluate, and deploy flows that link user prompts, actions through Python code, and calls to language learning models. Prompt flow is used in this architecture as the layer that orchestrates flows between the prompt, different data stores, and the language model.
  - [Managed online endpoints](#) let you deploy a flow for real-time inference. In this architecture, they're used as a PaaS endpoint for the chat UI to invoke the prompt flows hosted by Machine Learning.
- [Storage](#) is used to persist the prompt flow source files for prompt flow development.
- [Container Registry](#) lets you build, store, and manage container images and artifacts in a private registry for all types of container deployments. In this architecture, flows are packaged as container images and stored in Container Registry.
- [Azure OpenAI](#) is a fully managed service that provides REST API access to Azure OpenAI's language models, including the GPT-4, GPT-3.5-Turbo, and embeddings

set of models. In this architecture, in addition to model access, it's used to add common enterprise features such as [virtual network and private link](#), [managed identity](#) support, and content filtering.

- [Azure AI Search](#) is a cloud search service that supports [full-text search](#), [semantic search](#), [vector search](#), and [hybrid search](#). AI Search is included in the architecture because it's a common service used in the flows behind chat applications. AI Search can be used to retrieve and index data that's relevant for user queries. The prompt flow implements the RAG [Retrieval Augmented Generation](#) pattern to extract the appropriate query from the prompt, query AI Search, and use the results as grounding data for the Azure OpenAI model.

## Machine Learning prompt flow

The back end for enterprise chat applications generally follows a pattern similar to the following flow:

- The user enters a prompt in a custom chat user interface (UI).
- That prompt is sent to the back end by the interface code.
- The user intent, either question or directive, is extracted from the prompt by the back end.
- Optionally, the back end determines the data stores that hold data that's relevant to the user prompt
- The back end queries the relevant data stores.
- The back end sends the intent, the relevant grounding data, and any history provided in the prompt to the language model.
- The back end returns the result so that it can be displayed on the UI.

The back end could be implemented in any number of languages and deployed to various Azure services. This architecture uses Machine Learning prompt flow because it provides a [streamlined experience](#) to build, test, and deploy flows that orchestrate between prompts, back end data stores, and language models.

## Prompt flow runtimes

Machine Learning can directly host two types of prompt flow runtimes.

- **Automatic runtime:** A serverless compute option that manages the lifecycle and performance characteristics of the compute and allows flow-driven customization of the environment.
- **Compute instance runtime:** An always-on compute option in which the workload team must select the performance characteristics. This runtime offers more customization and control of the environment.

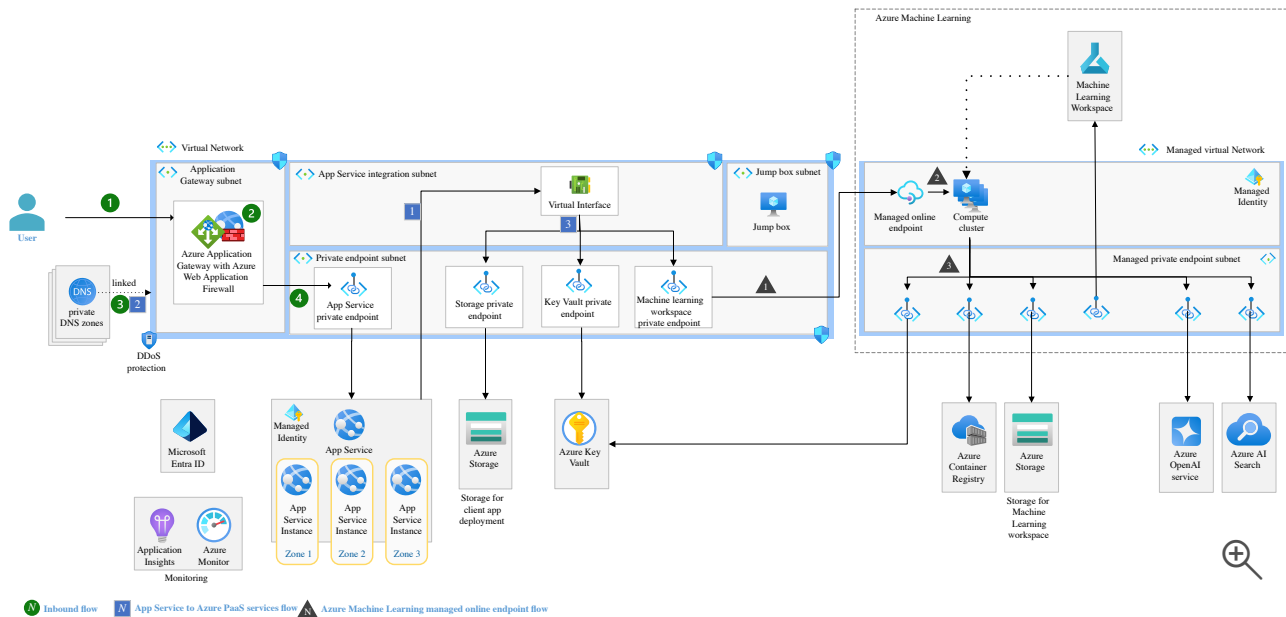
Prompt flows can also be hosted external to Machine Learning compute on host container host platforms. This architecture uses App Service to demonstrate external hosting.

## Networking

Along with identity-based access, network security is at the core of the baseline end-to-end chat architecture that uses OpenAI. From a high level, the network architecture ensures that:

- Only a single, secure entry point for chat UI traffic.
- Network traffic is filtered.
- Data in transit is encrypted end-to-end with Transport Layer Security (TLS).
- Data exfiltration is minimized by using Private Link to keep traffic in Azure.
- Network resources are logically grouped and isolated from each other through network segmentation.

## Network flows



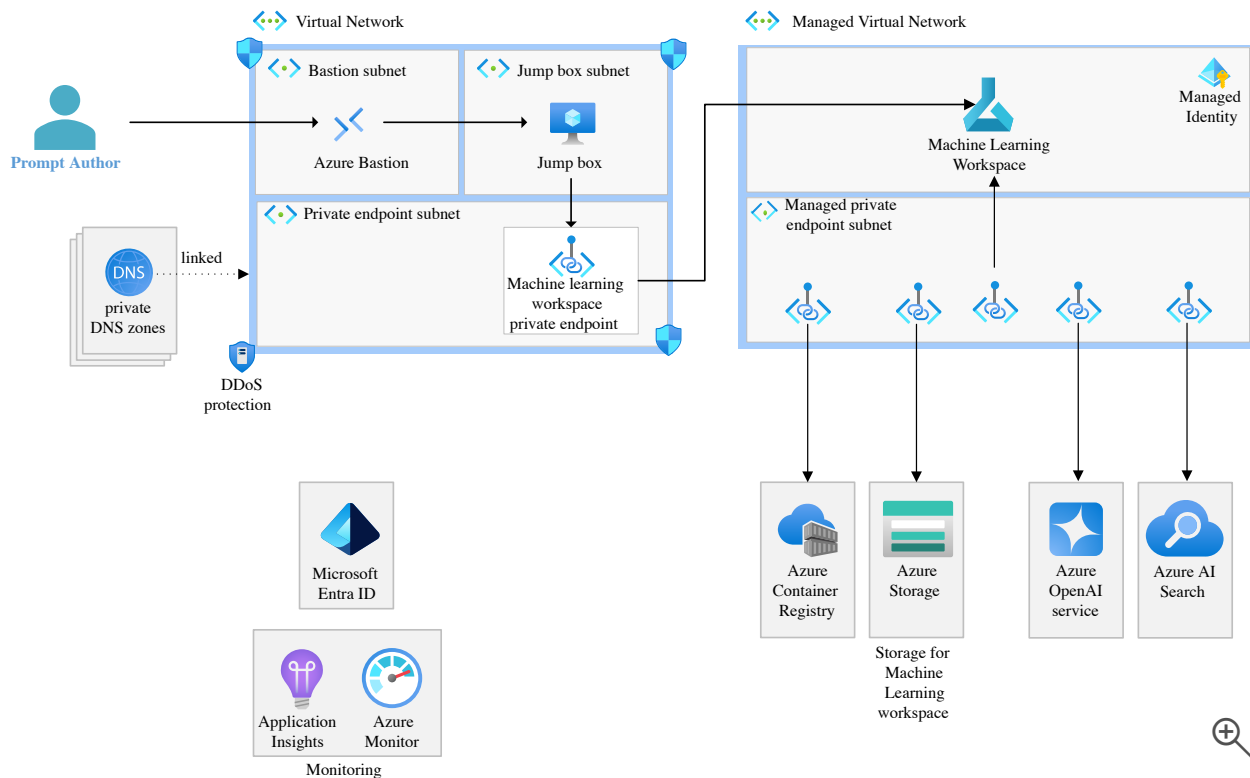
Two flows in this diagram are covered in the [baseline App Service web application architecture](#): The inbound flow from the end user to the chat UI (1) and the flow from App Service to [Azure PaaS services](#) (2). This section focuses on the Machine Learning online endpoint flow. The following flow goes from the chat UI that runs in the baseline App Service web application to the flow deployed to Machine Learning compute:

1. The call from the App Service-hosted chat UI is routed through a private endpoint to the Machine Learning online endpoint.
2. The online endpoint routes the call to a server running the deployed flow. The online endpoint acts as both a load balancer and a router.
3. Calls to Azure PaaS services required by the deployed flow are routed through managed private endpoints.

## Ingress to Machine Learning

In this architecture, public access to the Machine Learning workspace is disabled. Users can access the workspace via private access because the architecture follows the [private endpoint for the Machine Learning workspace](#) configuration. In fact, private endpoints are used throughout this architecture to complement identity-based security. For example, your App Service-hosted chat UI can connect to PaaS services that aren't exposed to the public internet, including Machine Learning endpoints.

Private endpoint access is also required for connecting to the Machine Learning workspace for flow authoring.



The diagram shows a prompt flow author connecting through Azure Bastion to a virtual machine jump box. From that jump box, the author can connect to the Machine Learning workspace through a private endpoint in the same network as the jump box. Connectivity to the virtual network could also be accomplished through ExpressRoute or VPN gateways and virtual network peering.

## Flow from the Machine Learning-managed virtual network to Azure PaaS services

We recommend that you configure the Machine Learning workspace for [managed virtual network isolation](#) that requires all outbound connections to be approved. This architecture follows that recommendation. There are two types of approved outbound rules. *Required outbound rules* are to resources required for the solution to work, such as

Container Registry and Storage. *User-defined outbound rules* are to custom resources, such as Azure OpenAI or AI Search, that your workflow is going to use. You must configure user-defined outbound rules. Required outbound rules are configured when the managed virtual network is created.


The outbound rules can be private endpoints, service tags, or fully qualified domain names (FQDNs) for external public endpoints. In this architecture, connectivity to Azure services such as Container Registry, Storage, Azure Key Vault, Azure OpenAI, and AI Search are connected through private link. Although not in this architecture, some common operations that might require configuring an FQDN outbound rule are downloading a pip package, cloning a GitHub repo, or downloading base container images from external repositories.

## Virtual network segmentation and security

The network in this architecture has separate subnets for the following purposes:

- Application Gateway
- App Service integration components
- Private endpoints
- Azure Bastion
- Jump box virtual machine
- Training - not used for model training in this architecture
- Scoring

Each subnet has a network security group (NSG) that limits both inbound and outbound traffic for those subnets to just what's required. The following table shows a simplified view of the NSG rules that the baseline adds to each subnet. The table provides the rule name and function.

 Expand table

Subnet	Inbound	Outbound



snet-appGateway	Allowances for our chat UI users source IPs (such as public internet), plus required items for the service.	Access to the App Service private endpoint, plus required items for the service.
snet-PrivateEndpoints	Allow only traffic from the virtual network.	Allow only traffic to the virtual network.
snet-AppService	Allow only traffic from the virtual network.	Allow access to the private endpoints and Azure Monitor.
AzureBastionSubnet	See guidance in <a href="#">Working with NSG access and Azure Bastion</a> .	See guidance in <a href="#">Working with NSG access and Azure Bastion</a> .
snet-jumpbox	Allow inbound Remote Desktop Protocol (RDP) and SSH from the Azure Bastion host subnet.	Allow access to the private endpoints
snet-agents	Allow only traffic from the virtual network.	Allow only traffic to the virtual network.
snet-training	Allow only traffic from the virtual network.	Allow only traffic to the virtual network.
snet-scoring	Allow only traffic from the virtual network.	Allow only traffic to the virtual network.

All other traffic is explicitly denied.

Consider the following points when implementing virtual network segmentation and security.

- Enable [DDoS Protection](#) for the virtual network with a subnet that's part of an

application gateway with a public IP address.

- [Add an NSG](#) to every subnet where possible. Use the strictest rules that enable full solution functionality.
- Use [application security groups](#) to group NSGs. Grouping NSGs makes rule creation easier for complex environments.

## Content filtering and abuse monitoring

Azure OpenAI includes a [content filtering system](#) that uses an ensemble of classification models to detect and prevent specific categories of potentially harmful content in both input prompts and output completions. Categories for this potentially harmful content include hate, sexual, self harm, violence, profanity, and jailbreak (content designed to bypass the constraints of a language model). You can configure the strictness of what you want to filter the content for each category, with options being low, medium, or high. This reference architecture adopts a stringent approach. Adjust the settings according to your requirements.

In addition to content filtering, the Azure OpenAI implements abuse monitoring features. Abuse monitoring is an asynchronous operation designed to detect and mitigate instances of recurring content or behaviors that suggest the use of the service in a manner that might violate the [Azure OpenAI code of conduct](#). You can request an [exemption of abuse monitoring and human review](#) if your data is highly sensitive or if there are internal policies or applicable legal regulations that prevent the processing of data for abuse detection.

## 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. They provide redundancy within a region for supporting services when two or more instances are deployed in across them. When one zone experiences downtime, the other zones within the region might still be unaffected. The architecture also ensures enough

instances of Azure services and configuration of those services to be spread across availability zones. For more information, see the [baseline](#) to review that guidance.

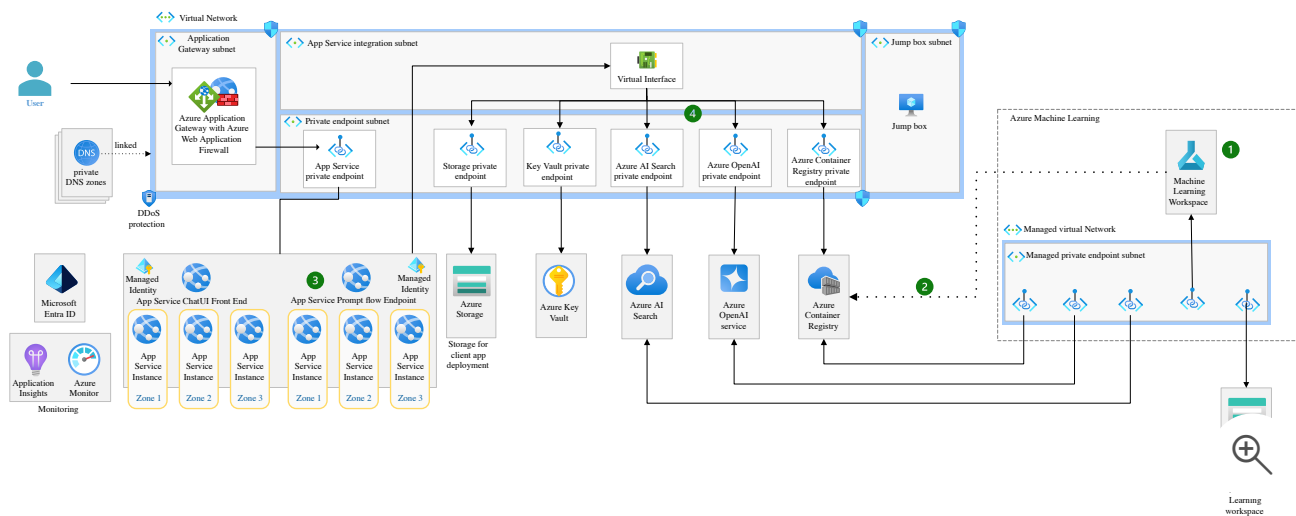
This section addresses reliability from the perspective of the components in this architecture not addressed in the App Service baseline, including Machine Learning, Azure OpenAI, and AI Search.

## Zonal redundancy for flow deployments

Enterprise deployments usually require zonal redundancy. To achieve zonal redundancy in Azure, resources must support [availability zones](#) and you must deploy at least three instances of the resource or enable the platform support when instance control isn't available. Currently, Machine Learning compute doesn't offer support for availability zones. To mitigate the potential impact of a datacenter-level catastrophe on Machine Learning components, it's necessary to establish clusters in various regions along with deploying a load balancer to distribute calls among these clusters. You can use health checks to help ensure that calls are only routed to clusters that are functioning properly.

Deploying prompt flows isn't limited to Machine Learning compute clusters. The executable flow, being a containerized application, can be deployed to any Azure service that's compatible with containers. These options include services like Azure Kubernetes Service (AKS), Azure Functions, Azure Container Apps, and App Service. Each of those services supports availability zones. To achieve zonal redundancy for prompt flow execution, without the added complexity of a multi-region deployment, you should deploy your flows to one of those services.

The following diagram shows an alternate architecture where prompt flows are deployed to App Service. App Service is used in this architecture because the workload already uses it for the chat UI and wouldn't benefit from introducing a new technology into the workload. Workload teams who have experience with AKS should consider deploying in that environment, especially if AKS is being used for other components in the workload.



The diagram is numbered for notable areas in this architecture:

1. Flows are still authored in Machine Learning prompt flow and the Machine Learning network architecture is unchanged. Flow authors still connect to the workspace authoring experience through a private endpoint, and the managed private endpoints are used to connect to Azure services when testing flows.
2. This dotted line indicates that containerized executable flows are pushed to Container Registry. Not shown in the diagram are the pipelines that containerize the flows and push to Container Registry.
3. There's another web app deployed to the same App Service plan that's already hosting the chat UI. The new web app hosts the containerized prompt flow, colocated on the same App Service plan that already runs at a minimum of three instances, spread across availability zones. These App Service instances connect to Container Registry over a private endpoint when loading the prompt flow container image.
4. The prompt flow container needs to connect to all dependent services for flow execution. In this architecture, the prompt flow container connects to AI Search and Azure OpenAI. PaaS services that were exposed only to the Machine Learning managed private endpoint subnet now also need to be exposed in the virtual network so that line of sight can be established from App Service.

# Azure OpenAI - reliability

Azure OpenAI doesn't currently support availability zones. To mitigate the potential impact of a datacenter-level catastrophe on model deployments in Azure OpenAI, it's necessary to deploy Azure OpenAI to various regions along with deploying a load balancer to distribute calls among the regions. You can use health checks to help ensure that calls are only routed to clusters that are functioning properly.

To support multiple instances effectively, we recommend that you externalize fine-tuning files, such as to a geo-redundant Storage account. This approach minimizes the state that's stored in the Azure OpenAI for each region. You must still fine tune files for each instance to host the model deployment.

It's important to monitor the required throughput in terms of tokens per minute (TPM) and requests per minute (RPM). Ensure that sufficient TPM is assigned from your quota to meet the demand for your deployments and prevent calls to your deployed models from being throttled. A gateway such as Azure API Management can be deployed in front of your OpenAI service or services and can be configured for retry if there are transient errors and throttling. API Management can also be used as a [circuit breaker](#) to prevent the service from getting overwhelmed with call, exceeding its quota.

# AI Search - reliability

Deploy AI Search with the Standard pricing tier or higher in a [region that supports availability zones](#), and deploy three or more replicas. The replicas automatically spread evenly across availability zones.

Consider the following guidance for determining the appropriate number of replicas and partitions:

- [Monitor AI Search](#).
- Use monitoring metrics and logs and performance analysis to determine the appropriate number of replicas to avoid query-based throttling and partitions and to avoid index-based throttling.

# Machine Learning - reliability

If you deploy to compute clusters behind the Machine Learning-managed online endpoint, consider the following guidance regarding scaling:

- [Automatically scale your online endpoints](#) to ensure enough capacity is available to meet demand. If usage signals aren't timely enough due to burst usage, consider overprovisioning to prevent an impact on reliability from too few instances being available.
- Consider creating scaling rules based on [deployment metrics](#) such as CPU load and [endpoint metrics](#) such as request latency.
- No less than three instances should be deployed for an active production deployment.
- Avoid deployments against in-use instances. Instead deploy to a new deployment and shift traffic over after the deployment is ready to receive traffic.

## ⓘ Note

The same [App Service scalability guidance](#) from the baseline architecture applies if you deploy your flow to App Service.

# Security

This architecture implements both a network and an identity security perimeter. From a network perspective, the only thing that should be accessible from the internet is the chat UI via Application Gateway. From an identity perspective, the chat UI should authenticate and authorize requests. Managed identities are used, where possible, to authenticate applications to Azure services.

This section describes identity and access management and security considerations for key rotation and Azure OpenAI model fine tuning.

# Identity and access management


The following guidance extends the [identity and access management guidance in the App Service baseline](#):

- Create separate managed identities for the following Machine Learning resources, where applicable:
  - Workspaces for flow authoring and management
  - Compute instances for testing flows
  - Online endpoints in the deployed flow if the flow is deployed to a managed online endpoint
- Implement identity-access controls for the chat UI by using Microsoft Entra ID

## Machine Learning role-based access roles

There are five [default roles](#) that you can use to manage access to your Machine Learning workspace: AzureML Data Scientist, AzureML Compute Operator, Reader, Contributor, and Owner. Along with these default roles, there's an AzureML Learning Workspace Connection Secrets Reader and an AzureML Registry User that can grant access to workspace resources such as the workspace secrets and registry.

This architecture follows the principle of least privilege by only assigning roles to the preceding identities where they're required. Consider the following role assignments.

 Expand table

Managed identity	Scope	Role assignments
Workspace managed identity	Resource group	Contributor
Workspace managed identity	Workspace Storage Account	Storage Blob Data Contributor

Workspace managed identity	Workspace Storage Account	Storage File Data Privileged Contributor
Workspace managed identity	Workspace Key Vault	Key Vault Administrator
Workspace managed identity	Workspace Container Registry	AcrPush
Online endpoint managed identity	Workspace Container Registry	AcrPull
Online endpoint managed identity	Workspace Storage Account	Storage Blob Data Reader
Online endpoint managed identity	Machine Learning workspace	AzureML Workspace Connection Secrets Reader
Compute instance managed identity	Workspace Container Registry	AcrPull
Compute instance managed identity	Workspace Storage Account	Storage Blob Data Reader

## Key rotation

There are two services in this architecture that use key-based authentication: Azure OpenAI and the Machine Learning managed online endpoint. Because you use key-based authentication for these services, it's important to:

- Store the key in a secure store, like Key Vault, for on-demand access from authorized clients, such as the Azure Web App hosting the prompt flow container.



- Implement a key rotation strategy. If you [manually rotate the keys](#), create a key expiration policy and use Azure Policy to monitor whether the key has been rotated.

## OpenAI model fine tuning

If you fine tune OpenAI models in your implementation, consider the following guidance:

- If you upload training data for fine tuning, consider using [customer-managed keys](#) for encrypting that data.
- If you store training data in a store such as Azure Blob Storage, consider using a customer-managed key for data encryption, a managed identity to control access to the data, and a private endpoint to connect to the data.

## Governance through policy

To help ensure alignment with security, consider using Azure Policy and network policy so that deployments align to the requirements of the workload. The use of platform automation through policy reduces the burden of manual validation steps and ensures governance even if pipelines are bypassed. Consider the following security policies:

- Disable key or other local authentication access in services like Azure AI services and Key Vault.
- Require specific configuration of network access rules or NSGs.
- Require encryption, such as the use of customer-managed keys.

## 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](#).

To see a pricing example for this scenario, use the [Azure pricing calculator](#) . You need to customize the example to match your usage because this example only includes the components included in the architecture. The most expensive components in the scenario are the chat UI and prompt flow compute and AI Search. Optimize those resources to save the most cost.

## Compute

Machine Learning prompt flow supports multiple options to host the executable flows. The options include managed online endpoints in Machine Learning, AKS, App Service, and Azure Kubernetes Service. Each of these options has their own billing model. The choice of compute affects the overall cost of the solution.

## Azure OpenAI

Azure OpenAI is a consumption-based service, and as with any consumption-based service, controlling demand against supply is the primary cost control. To do that in Azure OpenAI specifically, you need to use a combination of approaches:

- **Control clients.** Client requests are the primary source of cost in a consumption model, so controlling client behavior is critical. All clients should:
  - Be approved. Avoid exposing the service in such a way that supports free-for-all access. Limit access both through network and identity controls, such as keys or role-based access control (RBAC).
  - Be self-controlled. Require clients to use the token-limiting constraints offered by the API calls, such as `max_tokens` and `max_completions`.
  - Use batching, where practical. Review clients to ensure they're appropriately batching prompts.
  - Optimize prompt input and response length. Longer prompts consume more tokens, raising the cost, yet prompts that are missing sufficient context don't help the models yield good results. Create concise prompts that provide

enough context to allow the model to generate a useful response. Likewise, ensure that you optimize the limit of the response length.

- **Azure OpenAI playground** usage should be as necessary and on preproduction instances, so that those activities aren't incurring production costs.
- **Select the right AI model.** Model selection also plays a large role in the overall cost of Azure OpenAI. All models have strengths and weaknesses and are individually priced. Use the correct model for the use case to make sure that you're not overspending on a more expensive model when a less expensive model yields acceptable results. In this chat reference implementation, GPT 3.5-turbo was chosen over GPT-4 to save about an order of magnitude of model deployment costs while achieving sufficient results.
- **Understand billing breakpoints.** Fine-tuning is charged per-hour. To be the most efficient, you want to use as much of that time available per hour to improve the fine-tuning results while avoiding just slipping into the next billing period. Likewise, the cost for 100 images from image generation is the same as the cost for one image. Maximize the price break points to your advantage.
- **Understand billing models.** Azure OpenAI is also available in a commitment-based billing model through the [provisioned throughput](#) offering. After you have predictable usage patterns, consider switching to this prepurchase billing model if it's more cost effective at your usage volume.
- **Set provisioning limits.** Ensure that all provisioning quota is allocated only to models that are expected to be part of the workload, on a per-model basis. Throughput to already deployed models isn't limited to this provisioned quota while dynamic quota is enabled. Quota doesn't directly map to costs and that cost might vary.
- **Monitor pay-as-you-go usage.** If you use pay-as-you-go pricing, [monitor usage](#) of TPM and RPM. Use that information to inform architectural design decisions such as what models to use, and optimize prompt sizes.
- **Monitor provisioned throughput usage.** If you use [provisioned throughput](#),

monitor [provision-managed usage](#) to ensure that you aren't underusing the provisioned throughput that you purchased.

- **Cost management.** Follow the guidance on [using cost management features with OpenAI](#) to monitor costs, set budgets to manage costs, and create alerts to notify stakeholders of risks or anomalies.

## Operational excellence

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

## Machine Learning - built-in prompt flow runtimes

To minimize operational burden the **Automatic Runtime** is a serverless compute option within Machine Learning that simplifies compute management and delegates most of the prompt flow configuration to the running application's `requirements.txt` file and `flow.dag.yaml` configuration. This makes this choice low maintenance, ephemeral, and application-driven. Using **Compute Instance Runtime** or externalized compute, such as in this architecture, requires a more workload team-managed lifecycle of the compute, and should be selected when workload requirements exceed the configuration capabilities of the automatic runtime option.

## Monitoring

Diagnostics are configured for all services. All services but Machine Learning and App Service are configured to capture all logs. The Machine Learning diagnostics are configured to capture the audit logs that are all resource logs that record customer interactions with data or the settings of the service. App Service is configured to capture AppServiceHTTPLogs, AppServiceConsoleLogs, AppServiceAppLogs, and

AppServicePlatformLogs.

Evaluate building custom alerts for the resources in this architecture such as those found in the Azure Monitor baseline alerts. For example:

- [Container Registry alerts](#)
- [Machine Learning and Azure OpenAI alerts](#)
- [Azure Web Apps alerts](#)

## Language model operations

Deployment for Azure OpenAI-based chat solutions like this architecture should follow the guidance in [LLMOps with prompt flow with Azure DevOps](#) and [GitHub](#). Additionally, it must consider best practices for continuous integration and continuous delivery (CI/CD) and network-secured architectures. The following guidance addresses the implementation of flows and their associated infrastructure based on the LLMOps recommendations. This deployment guidance doesn't include the front-end application elements, which are unchanged from in the [Baseline highly available zone-redundant web application architecture](#).

## Development

Machine Learning prompt flow offers both a browser-based authoring experience in Machine Learning studio or through a [Visual Studio Code extension](#). Both options store the flow code as files. When you use Machine Learning studio, the files are stored in a Storage account. When you work in Microsoft Visual Studio Code, the files are stored in your local file system.

In order to follow [best practices for collaborative development](#), the source files should be maintained in an online source code repository such as GitHub. This approach facilitates tracking of all code changes, collaboration between flow authors and integration with [deployment flows](#) that test and validate all code changes.

For enterprise development, use the [Microsoft Visual Studio Code extension](#) and the [prompt flow SDK/CLI](#) for development. Prompt flow authors can build and test their

flows from Microsoft Visual Studio Code and integrate the locally stored files with the online source control system and pipelines. While the browser-based experience is well suited for exploratory development, with some work, it can be integrated with the source control system. The flow folder can be downloaded from the flow page in the Files panel, unzipped, and pushed to the source control system.

## Evaluation

Test the flows used in a chat application just as you test other software artifacts. It's challenging to specify and assert a single "right" answer for language model outputs, but you can use a language model itself to evaluate responses. Consider implementing the following automated evaluations of your language model flows:

- **Classification accuracy:** Evaluates whether the language model gives a "correct" or "incorrect" score and aggregates the outcomes to produce an accuracy grade.
- **Coherence:** Evaluates how well the sentences in a model's predicted answer are written and how they coherently connect with each other.
- **Fluency:** Assesses the model's predicted answer for its grammatical and linguistic accuracy.
- **Groundedness against context:** Evaluates how well the model's predicted answers are based on preconfigured context. Even if the language model responses are correct, if they can't be validated against the given context, then such responses aren't grounded.
- **Relevance:** Evaluates how well the model's predicted answers align with the question asked.

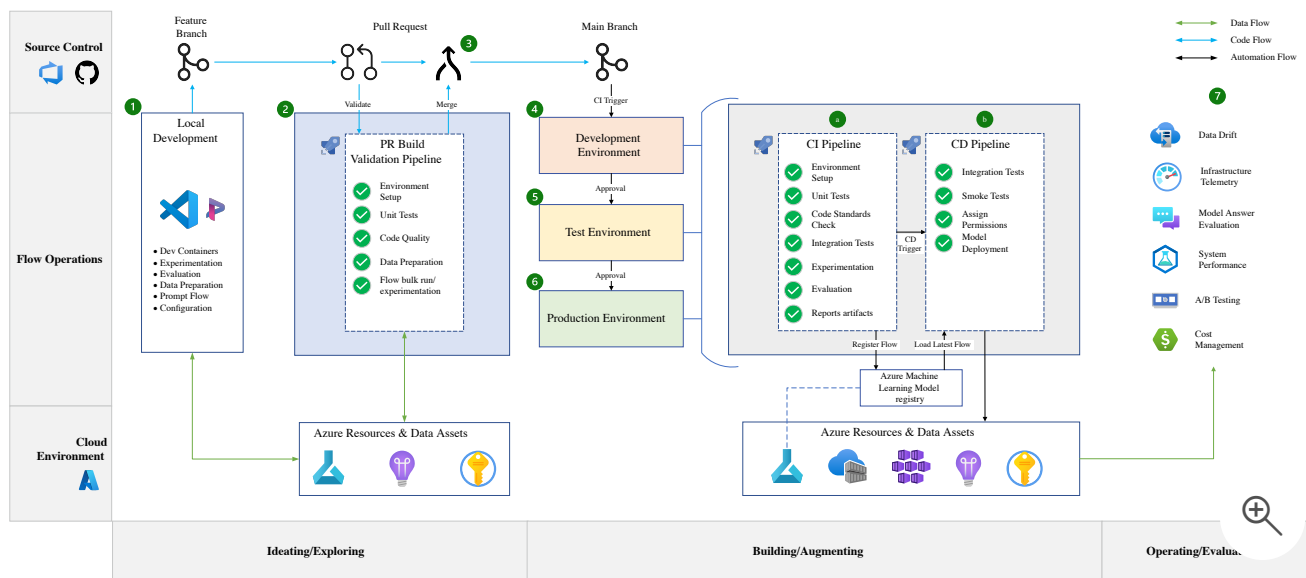
Consider the following guidance when implementing automated evaluations:

- Produce scores from evaluations and measure them against a predefined success threshold. Use these scores to report test pass/fail in your pipelines.
- Some of these tests require preconfigured data inputs of questions, context, and

ground truth.

- Include enough question-answer pairs to ensure the results of the tests are reliable, with at least 100-150 pairs recommended. These question-answer pairs are referred to as your "golden dataset." A larger population might be required depending on the size and domain of your dataset.
- Avoid using language models to generate any of the data in your golden dataset.

## Deployment Flow



1. The prompt engineer/data scientist opens a feature branch where they work on the specific task or feature. The prompt engineer/data scientist iterates on the flow using prompt flow for Microsoft Visual Studio Code, periodically committing changes and pushing those changes to the feature branch.
2. Once local development and experimentation are completed, the prompt engineer/data scientist opens a pull request from the Feature branch to the Main branch. The pull request (PR) triggers a PR pipeline. This pipeline runs fast quality checks that should include:
  - Execution of experimentation flows
  - Execution of configured unit tests

- Compilation of the codebase
  - Static code analysis
3. The pipeline can contain a step that requires at least one team member to manually approve the PR before merging. The approver can't be the committer and they must have prompt flow expertise and familiarity with the project requirements. If the PR isn't approved, the merge is blocked. If the PR is approved, or there's no approval step, the feature branch is merged into the Main branch.
  4. The merge to Main triggers the build and release process for the Development environment. Specifically:
    - a. The CI pipeline is triggered from the merge to Main. The CI pipeline performs all the steps done in the PR pipeline, and the following steps:
      - Experimentation flow
      - Evaluation flow
      - Registers the flows in the Machine Learning Registry when changes are detected
    - a. The CD pipeline is triggered after the completion of the CI pipeline. This flow performs the following steps:
      - Deploys the flow from the Machine Learning registry to a Machine Learning online endpoint
      - Runs integration tests that target the online endpoint
      - Runs smoke tests that target the online endpoint
  5. An approval process is built into the release promotion process – upon approval, the CI & CD processes described in steps 4.a. & 4.b. are repeated, targeting the Test environment. Steps a. and b. are the same, except that user acceptance tests are run after the smoke tests in the Test environment.
  6. The CI & CD processes described in steps 4.a. & 4.b. are run to promote the release to the Production environment after the Test environment is verified and approved.
  7. After release into a live environment, the operational tasks of monitoring



performance metrics and evaluating the deployed language models take place. This includes but isn't limited to:

- Detecting data drifts
- Observing the infrastructure
- Managing costs
- Communicating the model's performance to stakeholders

## Deployment guidance

You can use Machine Learning endpoints to deploy models in a way that enables flexibility when releasing to production. Consider the following strategies to ensure the best model performance and quality:

- Blue/green deployments: With this strategy, you can safely deploy your new version of the web service to a limited group of users or requests before directing all traffic over to the new deployment.
- A/B testing: Not only are blue/green deployments effective for safely rolling out changes, they can also be used to deploy new behavior that allows a subset of users to evaluate the impact of the change.
- Include linting of Python files that are part of the prompt flow in your pipelines. Linting checks for compliance with style standards, errors, code complexity, unused imports, and variable naming.
- When you deploy your flow to the network-isolated Machine Learning workspace, use a self-hosted agent to deploy artifacts to your Azure resources.
- The Machine Learning model registry should only be updated when there are changes to the model.
- The language models, the flows, and the client UI should be loosely coupled. Updates to the flows and the client UI can and should be able to be made without affecting the model and vice versa.

- When you develop and deploy multiple flows, each flow should have its own lifecycle, which allows for a loosely coupled experience when promoting flows from experimentation to production.

## Infrastructure

When you deploy the baseline Azure OpenAI end-to-end chat components, some of the services provisioned are foundational and permanent within the architecture, whereas other components are more ephemeral in nature, their existence tied to a deployment.

### Foundational components

Some components in this architecture exist with a lifecycle that extends beyond any individual prompt flow or any model deployment. These resources are typically deployed once as part of the foundational deployment by the workload team, and maintained apart from new, removed, or updates to the prompt flows or model deployments.

- Machine Learning workspace
- Storage account for the Machine Learning workspace
- Container Registry
- AI Search
- Azure OpenAI
- Azure Application Insights
- Azure Bastion
- Azure Virtual Machine for the jump box

### Ephemeral components

Some Azure resources are more tightly coupled to the design of specific prompt flows. This approach allows these resources to be bound to the lifecycle of the component and become ephemeral in this architecture. Azure resources are affected when the workload evolves, such as when flows are added or removed or when new models are introduced. These resources get re-created and prior instances removed. Some of these resources

are direct Azure resources and some are data plane manifestations within their containing service.

- The model in the Machine Learning model registry should be updated, if changed, as part of the CD pipeline.
- The container image should be updated in the container registry as part of the CD pipeline.
- A Machine Learning endpoint is created when a prompt flow is deployed if the deployment references an endpoint that doesn't exist. That endpoint needs to be updated to [turn off public access](#).
- The Machine Learning endpoint's deployments are updated when a flow is deployed or deleted.
- The key vault for the client UI must be updated with the key to the endpoint when a new endpoint is created.

## Performance efficiency

Performance efficiency is your workload's ability to efficiently scale to meet the demands placed on it by users. For more information, see [Design review checklist for Performance Efficiency](#).

This section describes performance efficiency from the perspective of Azure Search, Azure OpenAI, and Machine Learning.

### Azure Search - performance efficiency

Follow the guidance to [analyze performance in AI Search](#).

### Azure OpenAI - performance efficiency

- Determine whether your application requires [provisioned throughput](#) or the

shared hosting, or consumption, model. Provisioned throughput ensures reserved processing capacity for your OpenAI model deployments, which provides predictable performance and throughput for your models. This billing model is unlike the shared hosting, or consumption, model. The consumption model is best-effort and might be subject to noisy neighbor or other stressors on the platform.

- Monitor [provision-managed utilization](#) for provisioned throughput.

## Machine Learning - performance efficiency

If you deploy to Machine Learning online endpoints:

- Follow the guidance about how to [autoscale an online endpoint](#). Do this to remain closely aligned with demand without excessive overprovisioning, especially in low-usage periods.
- Choose the appropriate virtual machine SKU for the online endpoint to meet your performance targets. Test the performance of both lower instance count and bigger SKUs versus larger instance count and smaller SKUs to find an optimal configuration.

## Deploy this scenario

To deploy and run the reference implementation, follow the steps in the [OpenAI end-to-end baseline reference implementation](#) .

## Contributors

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

- [Rob Bagby](#) | Patterns & Practices - Microsoft
- [Freddy Ayala](#) | Cloud Solution Architect - Microsoft

- [Prabal Deb](#) | Senior Software Engineer - Microsoft
- [Raouf Aliouat](#) | Software Engineer II - Microsoft
- [Ritesh Modi](#) | Principal Software Engineer - Microsoft
- [Ryan Pfalz](#) | Senior Solution Architect - Microsoft

*To see non-public LinkedIn profiles, sign in to LinkedIn.*

## Next step

[Azure OpenAI](#)

## Related resources

- [Azure OpenAI](#)
- [Azure OpenAI language models](#)
- [Machine Learning prompt flow](#)
- [Workspace managed virtual network isolation](#)
- [Configure a private endpoint for a Machine Learning workspace](#)
- [Content filtering](#)

---

## Feedback

Was this page helpful?

 Yes

 No