Implementation Guide

Generative AI Application Builder on AWS

Generative AI Application Builder on AWS: Implementation Guide

Table of Contents

This solution facilitates the development, rapid experimentation, and deployment of generative artificial intelligence (AI) applications

Generative AI Application Builder on AWS facilitates the development, rapid experimentation, and deployment of generative artificial intelligence (AI) applications without requiring deep experience in AI. This AWS Solution accelerates development and streamlines experimentation by helping you:

- Ingest your business-specific data and documents
- Evaluate and compare the performance of large language models (LLMs)
- Run multi-step tasks and workflows with AI agents
- Rapidly build extensible applications, and deploy those applications with an enterprise-grade architecture

Generative AI Application Builder on AWS includes integrations with:

- LLMs available on Amazon Bedrock
- LLMs that you have deployed on Amazon SageMaker AI
- Amazon Bedrock Knowledge Bases for Retrieval-Augmented Generation (RAG)
- Amazon Bedrock Guardrails to implement safeguards and reduce hallucinations
- Amazon Bedrock Agents to build agentic workflows that can carry out task orchestrations and completion
- Amazon Bedrock AgentCore to build, deploy, and manage production-ready AI agents with extended runtime support
- Model Context Protocol (MCP) servers for enterprise data and tool integration

Additionally, this solution enables connections to your choice of model by using LangChain connectors. These connectors are available in an AWS Lambda function that deploys with the solution. You can start with the no-code deployment wizard to build generative AI applications for conversational search, AI- generated chatbots, text generation, and text summarization.

This implementation guide provides an overview of the Generative AI Application Builder on AWS solution, its reference architecture and components, considerations for planning the deployment, and configuration steps for deploying the solution to the Amazon Web Services (AWS) Cloud.

This guide is intended for solution architects, business decision makers, DevOps engineers, data scientists, and cloud professionals who want to implement Generative AI Application Builder on AWS in their environment.

Use this navigation table to quickly find answers to these questions:

If you want to . . .	Read . . .
Know the cost for running this solution. The estimated cost for running this solution varies based on the components you deploy and the number of queries. The cost to run the Deployment dashboard with default parameters and 100 active users in the US East (N.Virginia) Region for one month is approximately $20.12 USD per month. The cost for a Text use case deployed without RAG for 1 business user performing 100 queries per day with the LLM is approximately $12.39 USD per month. The cost for a RAG-enabled use case with an Amazon Kendra index supporting 8,000 interactions per day is approximately $204.26 USD per month, plus the cost of the knowledge base.	Cost
Understand the security considerations for this solution.	Security
Know how to plan for quotas for this solution.	Quotas

If you want to . . .	Read . . .
Know which AWS Regions support this solution.	Supported AWS Regions
View or download the AWS CloudForm ation template included in this solution to automatically deploy the infrastructure resources (the "stack") for this solution.	AWS CloudFormation template
Access the source code and optionally use the AWS Cloud Development Kit (AWS CDK) to deploy the solution.	GitHub repository

Features and benefits

The Generative AI Application Builder on AWS solution provides the following features:

Rapid experimentation

This solution allows users to experiment quickly by removing the heavy lifting required to deploy multiple instances with different configurations and compare outputs and performance. Experiment with multiple configurations of various LLMs, prompt engineering, enterprise knowledge bases, guardrails, AI agents, and other parameters.

Choice and configurability

With pre-built connectors to a variety of LLMs, such as models available through Amazon Bedrock, this solution gives you the flexibility to deploy the model of your choice, as well as the AWS and leading FM services you prefer. You can also enable Amazon Bedrock Agents to fulfill various tasks and workflows.

Agent Builder

Build and deploy production-ready AI agents with full lifecycle management. Configure system prompts, integrate Model Context Protocol (MCP) servers for enterprise tools and data access, and enable memory capabilities for context retention across conversations. Agents are deployed on Amazon Bedrock AgentCore with extended runtime support and real-time streaming responses.

Workflow Builder

Orchestrate multiple Agent Builder agents into complex workflows using hierarchical delegation. Create a supervisor agent that autonomously selects and coordinates specialized Agent Builder agents to handle multi-step tasks. Configure agent descriptions, delegation strategies, and workflow-level memory while reusing existing Agent Builder deployments.

Production-ready

Built with AWS Well-Architected design principles, this solution offers enterprise-grade security and scalability with high availability and low latency, ensuring seamless integration into your applications with high performance standards.

Extensible modular architecture

Extend this solution's functionality by integrating your existing projects or natively connecting additional AWS services. Because this is an open-source application, you can use the included LangChain orchestration layer or Lambda functions to connect with the services of your choice.

Integration with Service Catalog AppRegistry and Application Manager, a capability of AWS Systems Manager

This solution includes a Service Catalog AppRegistry resource to register the solution's CloudFormation template and its underlying resources as an application in both AWS Service Catalog AppRegistry and AWS Systems Manager Application Manager. With this integration, you can centrally manage the solution's resources.

Agent Builder vs Bedrock Agent use case

This solution provides two distinct approaches for working with AI agents, each suited for different use cases and requirements:

Feature	Bedrock Agent use case	Agent Builder
Purpose	Invoke pre-deployed Amazon Bedrock Agents	Build, deploy, and manage custom agents
Configuration	Agent ID and Alias ID only	Full agent configuration: system prompts, models, MCP servers, memory

Feature	Bedrock Agent use case	Agent Builder
Deployment	Simple invocation layer	Complete agent lifecycle on AgentCore Runtime
Runtime	Amazon Bedrock Agents service	Amazon Bedrock AgentCore with Strands SDK
Tool Integration	Configured in Bedrock Agents console	Model Context Protocol (MCP) servers and built in Strands tools
Memory	Managed by Bedrock Agents (up to 30 days)	AgentCore Memory with configura ble short-term and long-term retention
Customization	Limited to pre-deployed agent settings	Full control over prompts, models, tools, and behavior
Best for	Quick deployment of existing agents	Custom agent development and production deployments

Note

Both options support real-time streaming, conversation history, and enterprise-grade security.

Workflow Builder

Workflow Builder enables multi-agent orchestration by creating a supervisor agent that delegates work to specialized Agent Builder agents. Each workflow consists of:

- **Supervisor Agent**: The entrypoint agent that receives user requests and coordinates specialized agents
- **Specialized Agents**: Agent Builder use cases that the supervisor can delegate tasks to
- **Agents as Tools Pattern**: The supervisor registers each Agent Builder agent as a tool and autonomously selects which agents to use

Feature	Agent Builder	Workflow Builder
Purpose	Build and deploy single custom agents	Orchestrate multiple Agent Builder agents
Agent Type	Single agent with MCP tools	Supervisor agent + multiple Agent Builder agents
Tool Integration	MCP servers and Strands tools	Agent Builder agents registered as tools
Delegation	Direct tool invocation	Autonomous agent selection and delegation
Complexity	Single-agent tasks	Multi-step, multi-agent workflows
Agent Reuse	N/A	Reuses existing Agent Builder deployments
Best for	Focused, single-domain tasks	Complex workflows requiring multiple specializations

Note

- Workflows require at least 1 Agent Builder use case as a specialized agent agent
- All specialized agents must be Agent Builder use cases deployed in GAAB

Use cases

Question answering over enterprise data

LLMs and other foundation models have been pre-trained on a large corpus of data enabling them to perform well at many natural language processing (NLP) tasks. But most foundation models and LLMs are static and have been pre-trained, limiting their ability to accurately answer questions on topics which are either new, specialized, or proprietary. Using prompt-based learning, you can

leverage the powerful NLP and text generation features of an LLM to provide richer customer experiences over your enterprise data.

Rapid generative AI prototyping

Out of the box, the solution comes bundled with various model providers and use cases. With an easy to use deployment wizard, customers can deploy pre-built use cases to enable the rapid experimentation of different generative AI prototypes and workloads.

Multi LLM comparison and experimentation

LLMs perform differently, and given your application's specific needs, you may find that one LLM suits your application better than another. This may be for reasons related to performance, accuracy, cost, creativity, or many other factors. This solution lets you quickly deploy multiple use cases enabling you to experiment with and compare different configurations until you've found what meets your needs.

Concepts and definitions

This section describes key concepts and defines terminology specific to this solution:

admin user

Within the context of this guide, the admin user is the one responsible for managing the content contained within the deployment. This user gets access to the Deployment dashboard UI and is primarily responsible for curating the business user experience. This is our primary target customer.

business user

Within the context of this guide, the business user represents the individuals who the use case has been deployed for. They are the consumers of the knowledge base and the customer responsible for evaluating and experimenting with the LLMs.

Deployment dashboard

The Deployment dashboard is a web interface that serves as a management console for admin users to view, manage, and create their *use cases*. This dashboard enables customers to rapidly experiment, iterate, and productionize various AI/ML workloads leveraging LLMs.

DevOps user

Within the context of this guide, the DevOps user is the one responsible for deploying the solution within the AWS account and for managing the infrastructure, updating the solution, monitoring performance, and maintaining the overall health and lifecycle of the solution.

use case

Use cases are isolated applications from the overall solution which integrate with LLMs to enable richer customer experiences by enabling the addition of a natural language interface into new or existing applications. Use cases are deployable through the Deployment dashboard or on their own.

> **Note**
>
> For a general reference of AWS terms, see the AWS Glossary.

Architecture overview

This section provides reference implementation architecture diagrams for the components deployed with this solution.

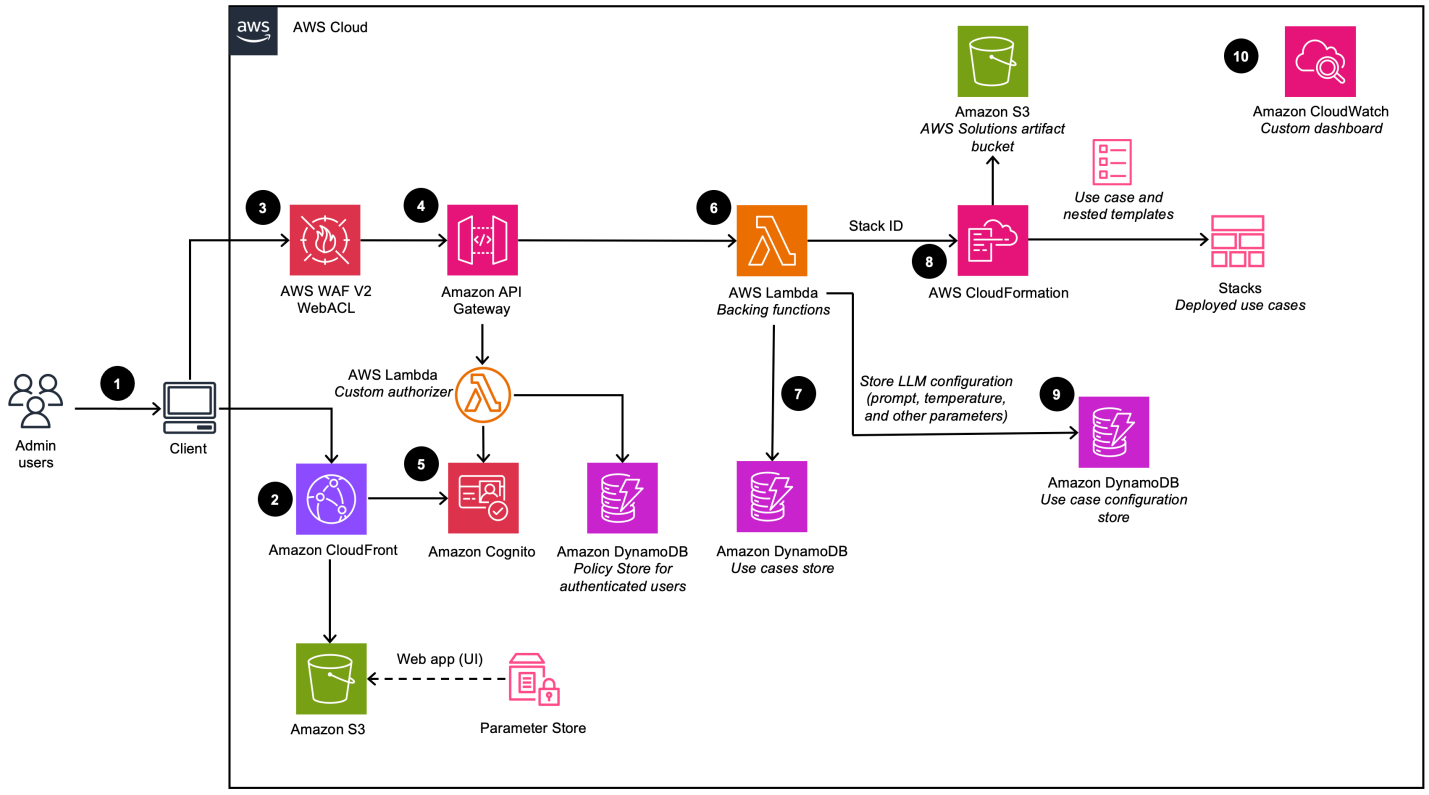
Architecture diagrams

To support multiple use cases and business needs, this solution provides six AWS CloudFormation templates:

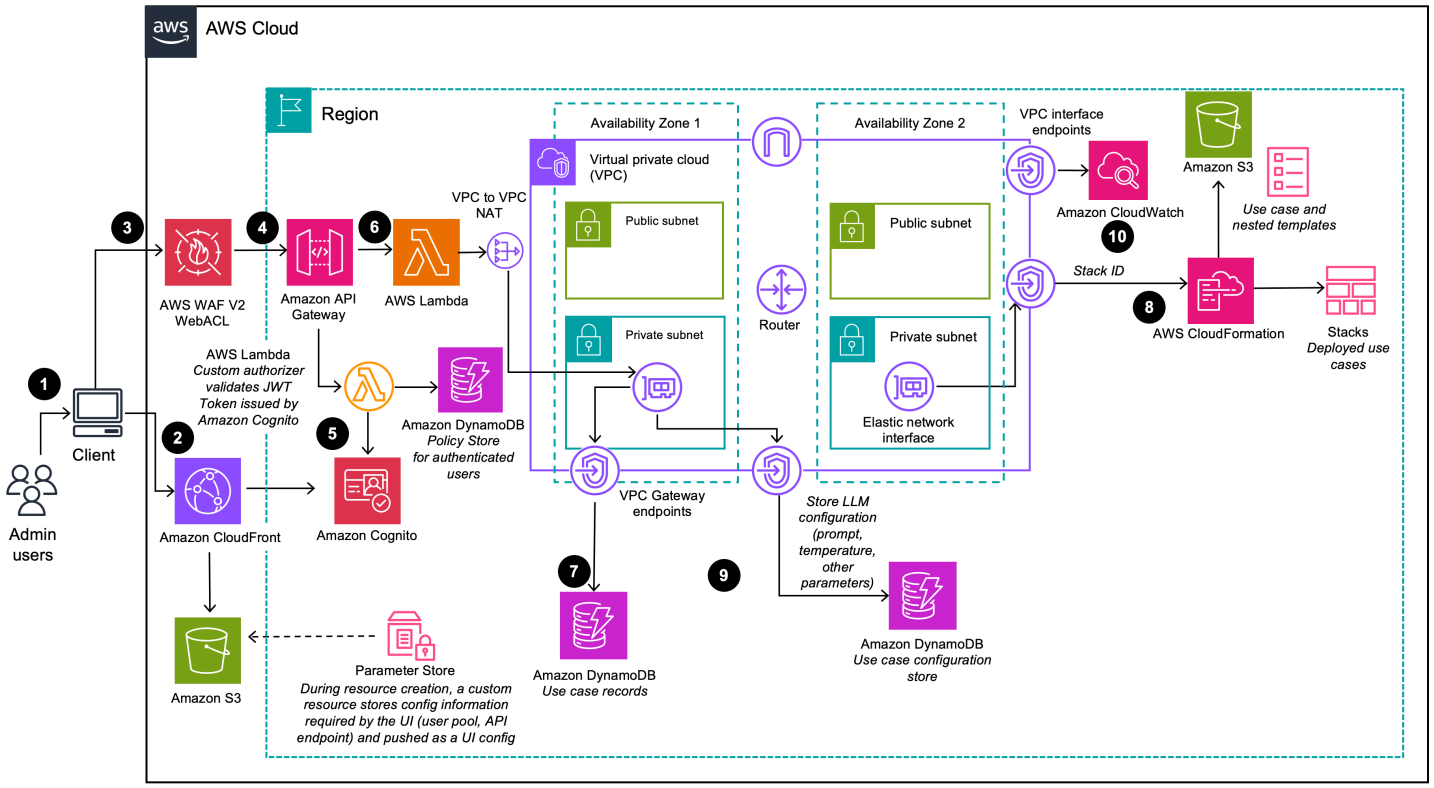
1. **Deployment dashboard** - The Deployment dashboard is a web interface that serves as a management console for admin users to view, manage, and create their use cases. This dashboard enables customers to rapidly experiment, iterate, and productionize various AI/ML workloads leveraging LLMs.
2. **Text use case** - The Text use case enables users to experience a natural language interface using generative AI. This use case can be integrated into new or existing applications, and is deployable through the Deployment dashboard or independently through a provided URL.
3. **Bedrock Agent use case** - The Bedrock Agent use case enables the use of existing Bedrock Agents to complete tasks or automate repeated workflows.
4. **MCP Server** - The MCP Server use case enables deployment and management of Model Context Protocol servers that provide standardized tool and resource access to AI applications. Supports both gateway methods for wrapping existing Lambda functions, APIs, and external MCP servers, and runtime methods for deploying custom containerized MCP servers.
5. **Agent Builder** - The Agent Builder enables creation and deployment of production-ready AI agents on Amazon Bedrock AgentCore with full configuration control, MCP server integration, and memory management capabilities.
6. **Workflow Builder** - The Workflow Builder enables creation of supervisor agents that orchestrate multiple Agent Builder agents using the Agents as Tools delegation pattern for complex multi-agent workflows.

Deployment dashboard

Depicts Deployment dashboard architecture (when deployed with VPC option disabled)



Depicts Deployment dashboard architecture (when deployed with VPC option enabled)



Note

AWS CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs.

The high-level process flow for the solution components deployed with the AWS CloudFormation template is as follows:

1. Admin users log in to the Deployment Dashboard user interface (UI).
2. Amazon CloudFront delivers the web UI, which is hosted in an Amazon Simple Storage Service (Amazon S3) bucket.
3. AWS WAF protects the APIs from attacks. This solution configures a set of rules called a web access control list (web ACL) that allows, blocks, or counts web requests based on configurable, user defined web security rules and conditions.
4. The web UI leverages a set of REST APIs that are exposed using Amazon API Gateway.
5. Amazon Cognito authenticates users and backs both the CloudFront web UI and API Gateway.
6. AWS Lambda provides the business logic for the REST endpoints. This *backing* Lambda function manages and creates the necessary resources to perform use case deployments using AWS CloudFormation.
7. Amazon DynamoDB stores the list of deployments.
8. When a new use case is created by the admin user, the *backing* Lambda function initiates a CloudFormation stack creation event for the requested use case.
9. All of the LLM configuration options provided by the admin user in the deployment wizard are saved in DynamoDB. The deployment uses this DynamoDB table to configure the LLM at runtime.
10. Using Amazon CloudWatch, this solution collects operational metrics from various services to generate custom dashboards that allow you to monitor the solution's performance and operational health.

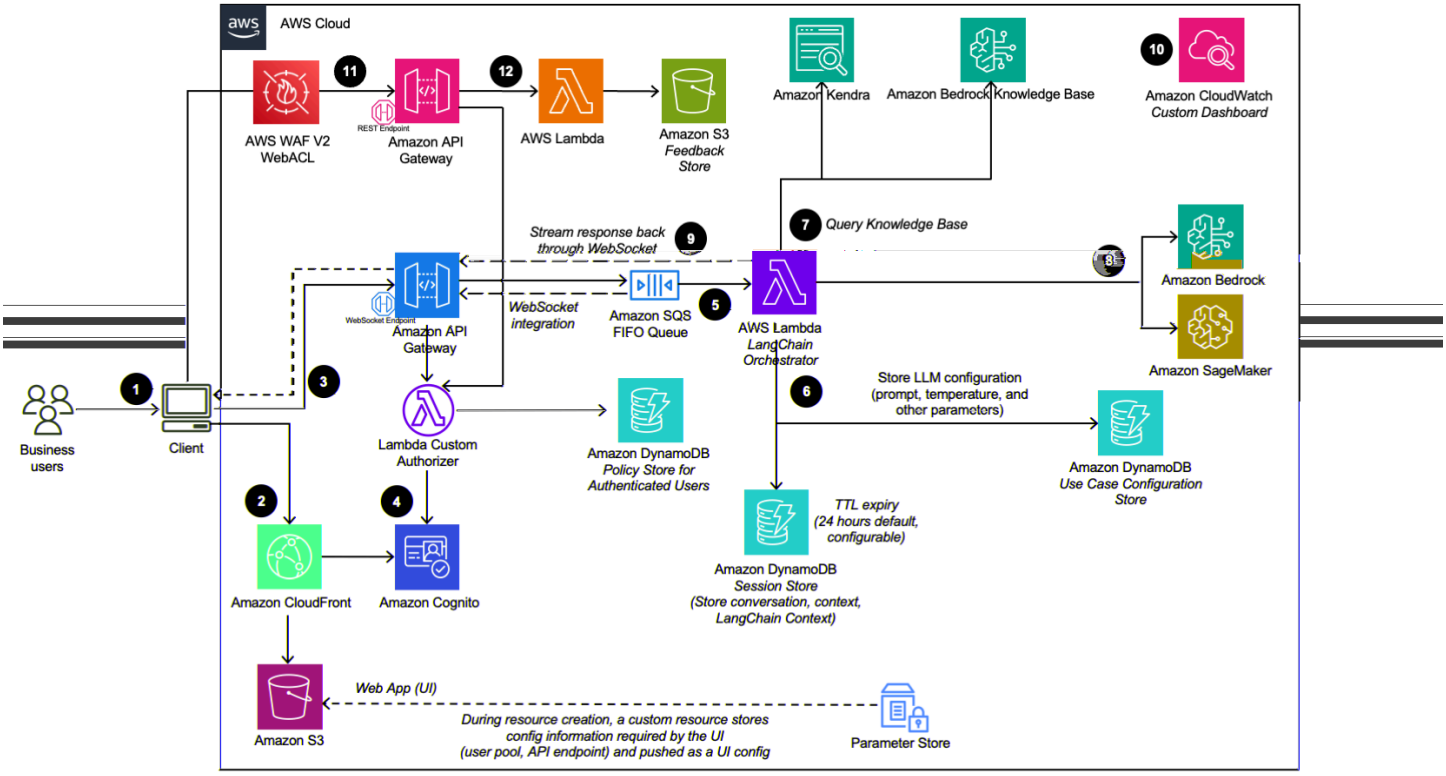
Note

- If you choose to deploy this solution in an Amazon VPC, the data will be routed within your private network.

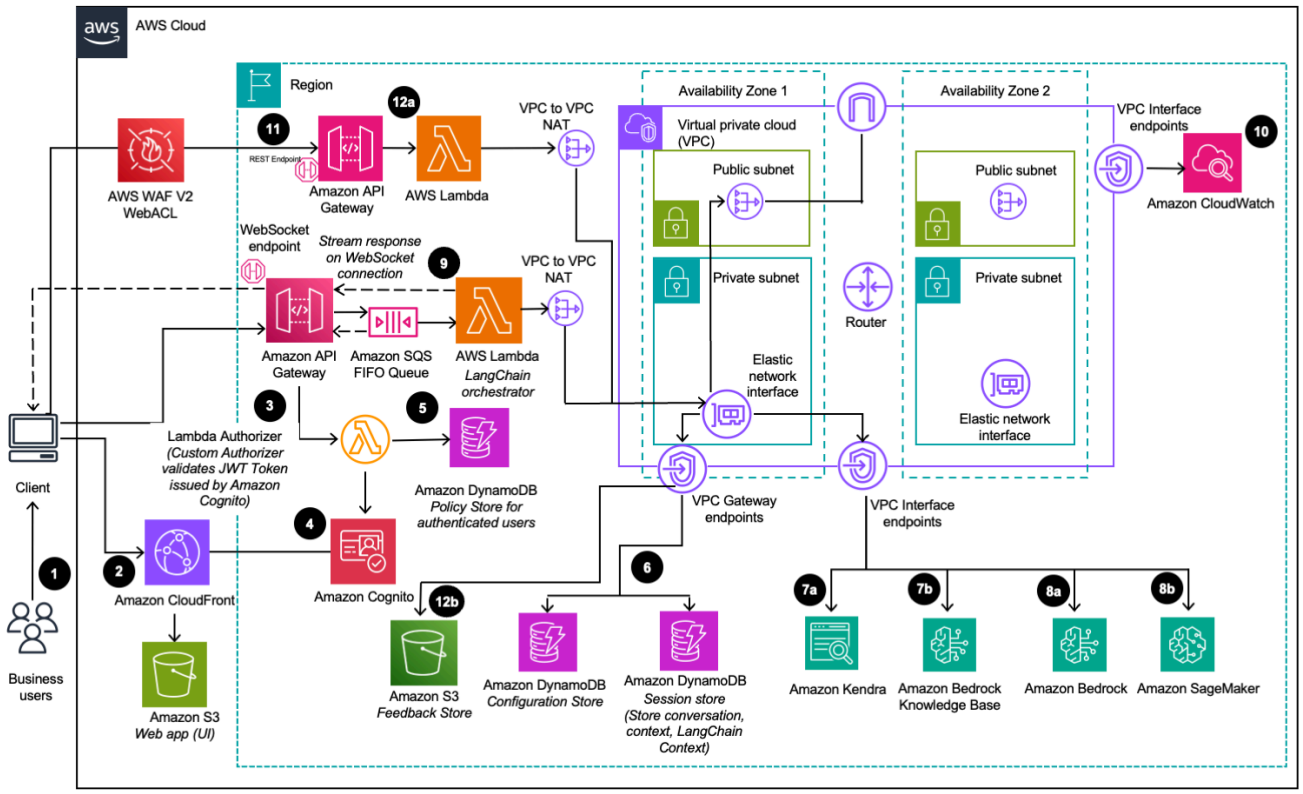
- Although the Deployment dashboard can be launched in most AWS Regions, the deployed use cases have certain restrictions based on service availability. See Supported AWS Regions for more details.

Text use case

Depicts Text use case architecture (when deployed with VPC option disabled)



Depicts Text use case architecture (when deployed with VPC option enabled)



The high-level process flow for the solution components deployed with the AWS CloudFormation template is as follows:

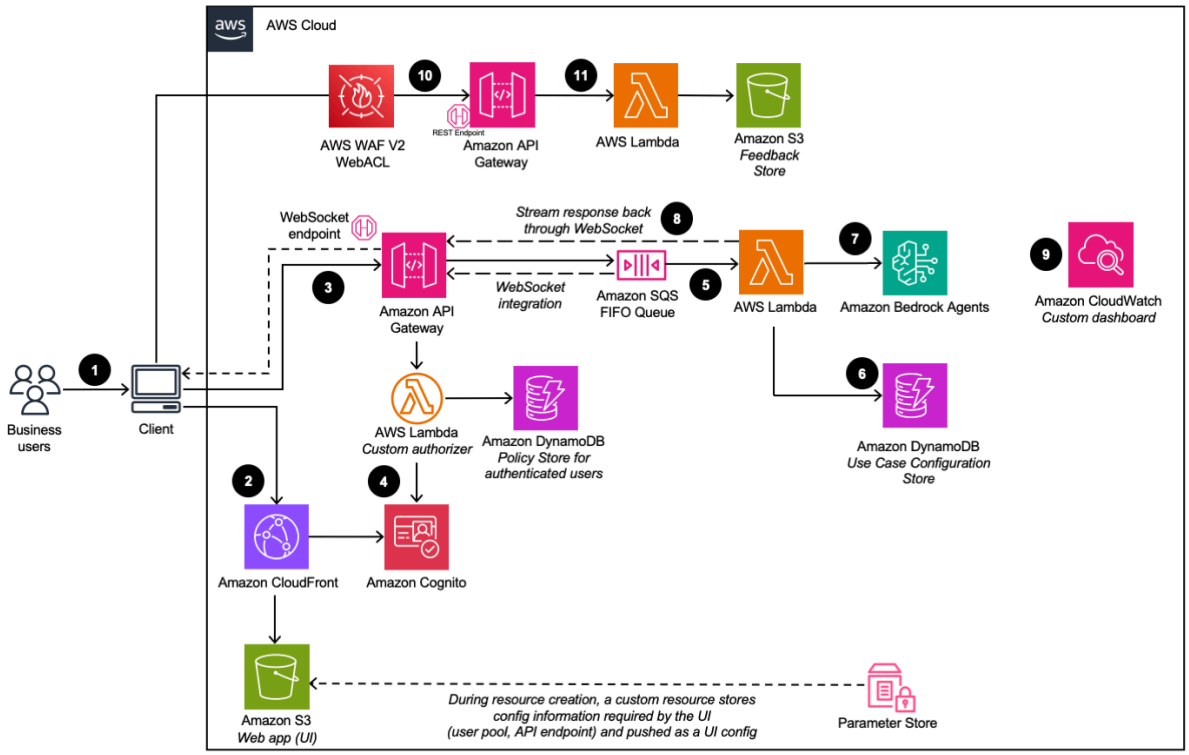
1. Admin users deploy the use case using the Deployment Dashboard. Business users log in to the use case UI.
2. CloudFront delivers the web UI which is hosted in an S3 bucket.
3. The web UI leverages a WebSocket integration built using API Gateway. The API Gateway is backed by a custom Lambda authorizer function, which returns the appropriate AWS Identity and Access Management (IAM) policy based on the Amazon Cognito group the authenticating user belongs to. The policy is stored in DynamoDB.
4. Amazon Cognito authenticates users and backs both the CloudFront web UI and API Gateway.
5. Incoming requests from the business user are passed from API Gateway to an Amazon SQS queue and then to the *LangChain Orchestrator*. The *LangChain Orchestrator* is a collection of Lambda functions and layers that provide the business logic for fulfilling requests coming from the business user. The queue enables the asynchronous operation of the API Gateway to Lambda integration. The queue passes connection information to the Lambda functions which will then post results directly back to the API Gateway websocket connection to support long running inference calls.

6. The *LangChain Orchestrator* uses Amazon DynamoDB to get the configured LLM options and necessary session information (such as the chat history).
7. If the deployment has a knowledge base enabled, then the *LangChain Orchestrator* leverages Amazon Kendra or Knowledge Bases for Amazon Bedrock to run a search query to retrieve document excerpts.
8. Using the chat history, query, and context from the knowledge base, the *LangChain Orchestrator* creates the final prompt and sends the request to the LLM hosted on Amazon Bedrock or Amazon SageMaker AI.
9. When the response comes back from the LLM, the *LangChain Orchestrator* streams the response back through the API Gateway WebSocket to be consumed by the client application.
10. Using Amazon CloudWatch, this solution collects operational metrics from various services to generate custom dashboards that allow you to monitor the deployment's performance and operational health.
11. If feedback collection is enabled, a REST API endpoint, leveraging Amazon API Gateway is made available for the collection of user feedback.
12. The feedback backing lambda, augments the submitted feedback with additional use case specific metadata (e.g. model used) and stores the data in Amazon S3 for later analysis and reporting by the DevOps users.

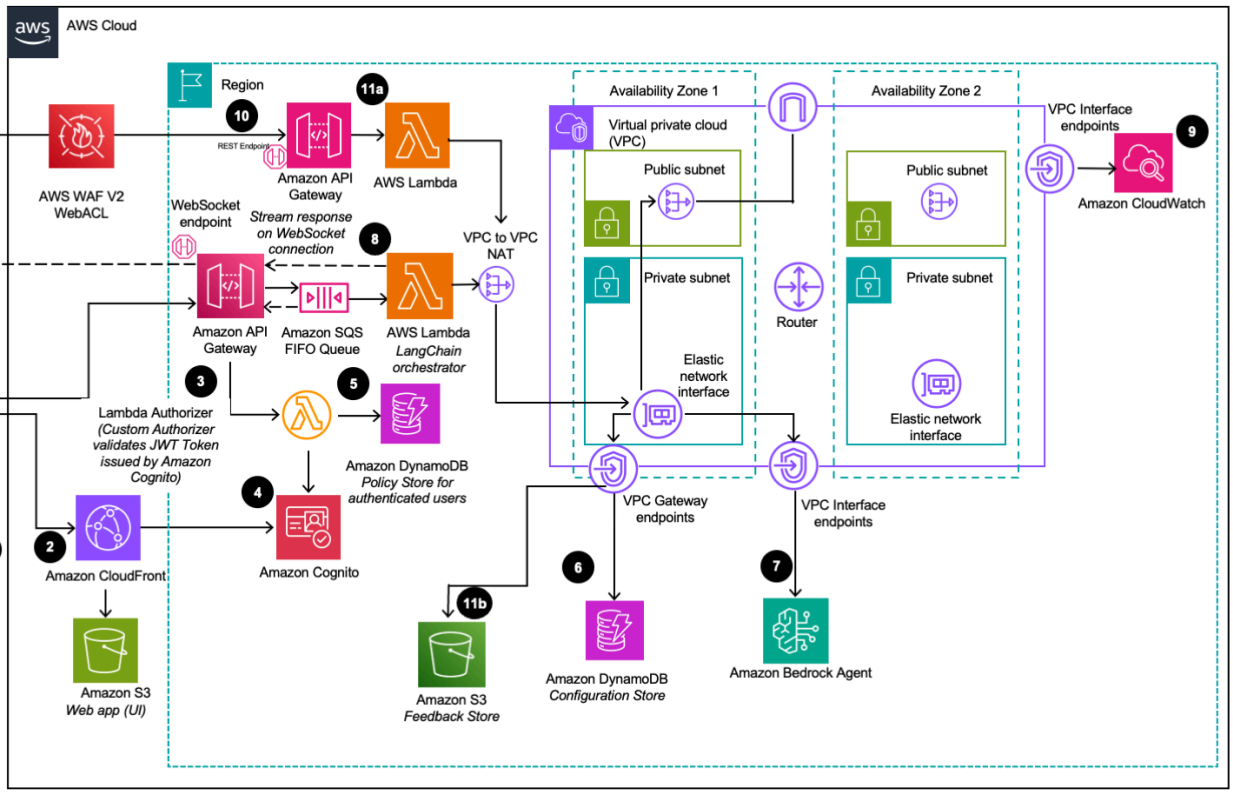
> **Note**
>
> If you choose to deploy this solution in an Amazon VPC, the data will be routed to your private network.

Bedrock Agent use case

Depicts Bedrock Agent use case architecture (when deployed with VPC option disabled)



Depicts Bedrock Agent use case architecture (when deployed with VPC option enabled)



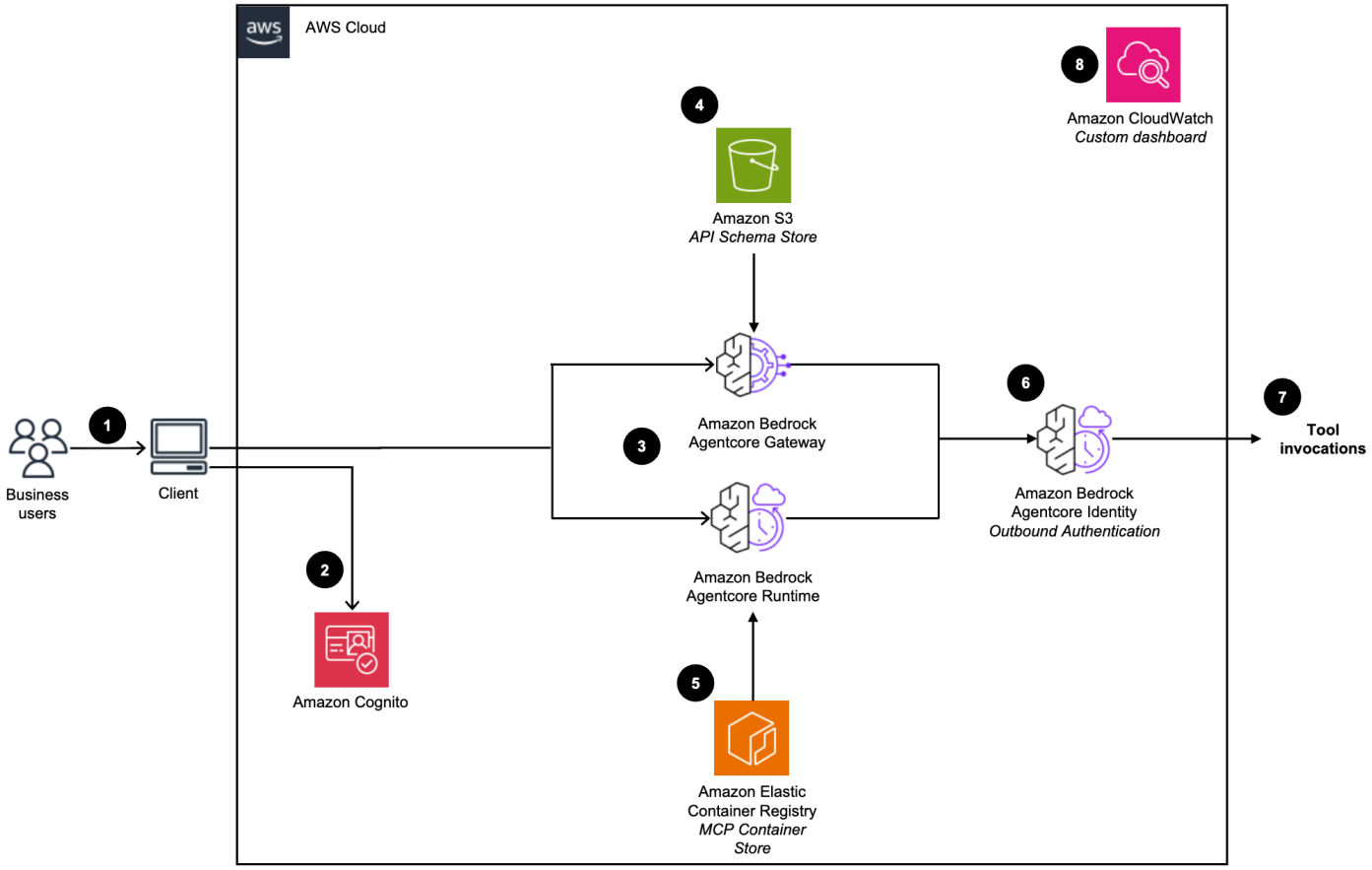
The high-level process flow for the solution components deployed with the AWS CloudFormation template is as follows:

1. Admin users deploy the use case using the Deployment Dashboard. Business users sign in to the use case UI.
2. CloudFront delivers the web UI which is hosted in an S3 bucket.
3. The web UI leverages a WebSocket integration built using API Gateway. The API Gateway is backed by a custom Lambda authorizer function, which returns the appropriate AWS Identity and Access Management(IAM) policy based on the Amazon Cognito group the authenticating user belongs to. The policy is stored in DynamoDB.
4. Amazon Cognito authenticates users and backs both the CloudFront web UI and API Gateway.
5. Incoming requests from the business user are passed from API Gateway to an Amazon SQS queue and then to the AWS Lambda function. The queue enables the asynchronous operation of the API Gateway to Lambda integration. The queue passes connection information to the Lambda function which will then post results directly back to the API Gateway websocket connection to support long running inference calls.
6. The AWS Lambda function uses Amazon DynamoDB to get the use case configurations as needed
7. Using the user input and any relevant use case configurations, the AWS Lambda function builds and sends a request payload to the configured Amazon Bedrock Agent to fulfill the user intent.
8. When the response comes back from the Amazon Bedrock Agent, the Lambda function streams the response back through the API Gateway WebSocket to be consumed by the client application.
9. Using Amazon CloudWatch, this solution collects operational metrics from various services to generate custom dashboards that allow you to monitor the deployment's performance and operational health.
10. If feedback collection is enabled, a REST API endpoint, leveraging Amazon API Gateway is made available for the collection of user feedback.
11. The feedback backing lambda, augments the submitted feedback with additional use case specific metadata and stores the data in Amazon S3 for later analysis and reporting by the DevOps users.

> **Note**
>
> If you choose to deploy this solution in an Amazon VPC, data will be routed within your private network.

MCP Server use case

Depicts MCP Server use case architecture



The MCP Server use case enables deployment and management of Model Context Protocol servers on Amazon Bedrock AgentCore. MCP servers provide a standardized interface for AI applications to access tools, resources, and enterprise data sources.

The solution supports two deployment methods:

- **Gateway method**: Wraps existing Lambda functions, REST APIs, or external MCP servers as MCP tools, handling protocol translation automatically

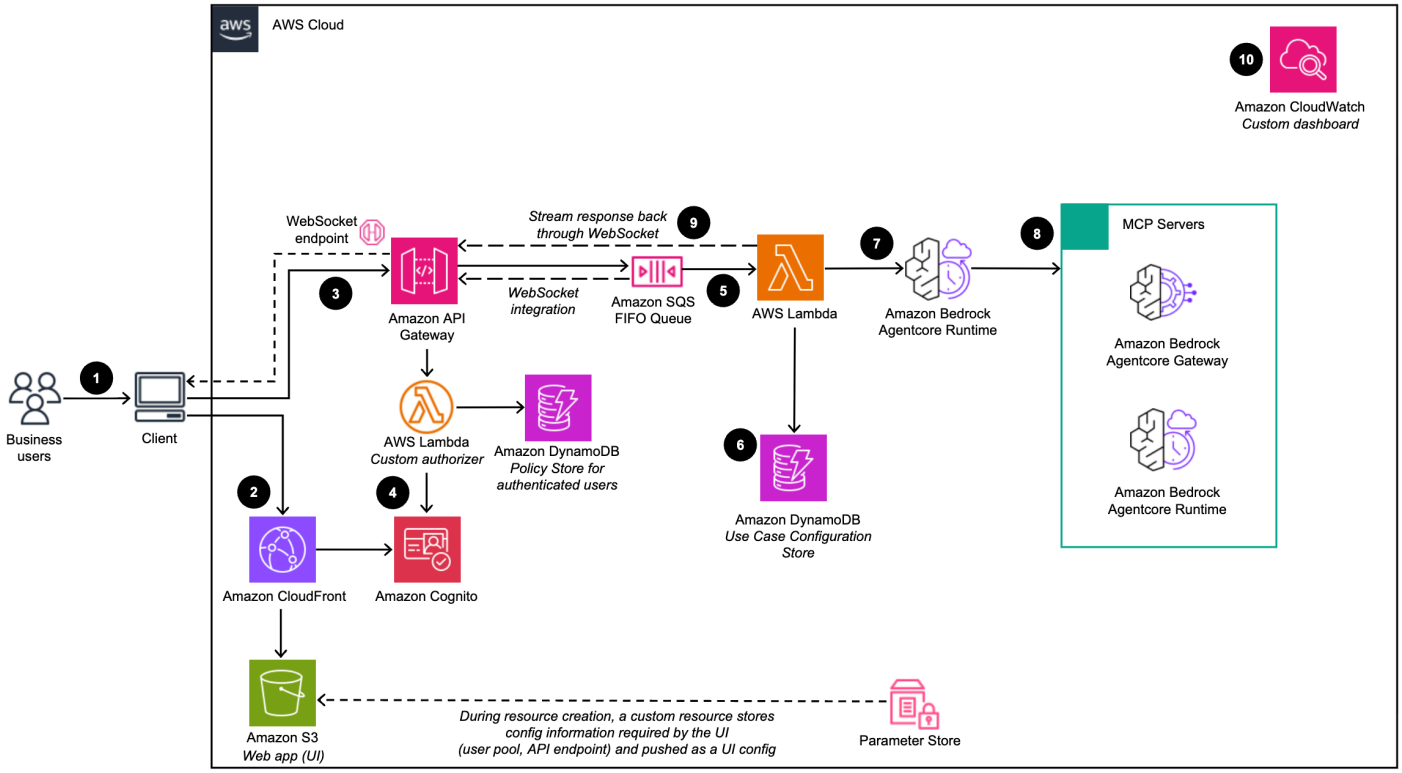
- **Runtime method**: Deploys custom containerized MCP servers from Amazon ECR images

The high-level process flow for MCP Server deployment is as follows:

1. Admin users deploy the MCP Server use case using the Deployment Dashboard, selecting either Gateway or Runtime deployment method.
2. This action is authenticated with Amazon Cognito.
3. For the Gateway deployment, the solution creates an Amazon Bedrock AgentCore Gateway that transforms existing Lambda functions, APIs, or external MCP servers into MCP-compliant tools. For the Runtime deployment, the solution deploys containerized MCP servers on Amazon Bedrock AgentCore Runtime using provided ECR images.
4. Gateway deployments retrieve the necessary API/Lambda/Smithy schemas from their uploaded location in Amazon S3, or connect directly to MCP Server URL endpoints.
5. Runtime deployments retrieve the containerized MCP server provided by the user from Amazon Elastic Container Registry (ECR)
6. The MCP Server is instrumented with an Amazon Bedrock AgentCore Identity OAuth client
7. The MCP Server makes the associated tools available at the /mcp endpoint for Agents to discover.
8. Amazon CloudWatch collects operational metrics and logs from MCP server deployments for monitoring and troubleshooting.

Agent Builder use case

Depicts Agent Builder architecture



The high-level process flow for the Agent Builder components deployed with the AWS CloudFormation template is as follows:

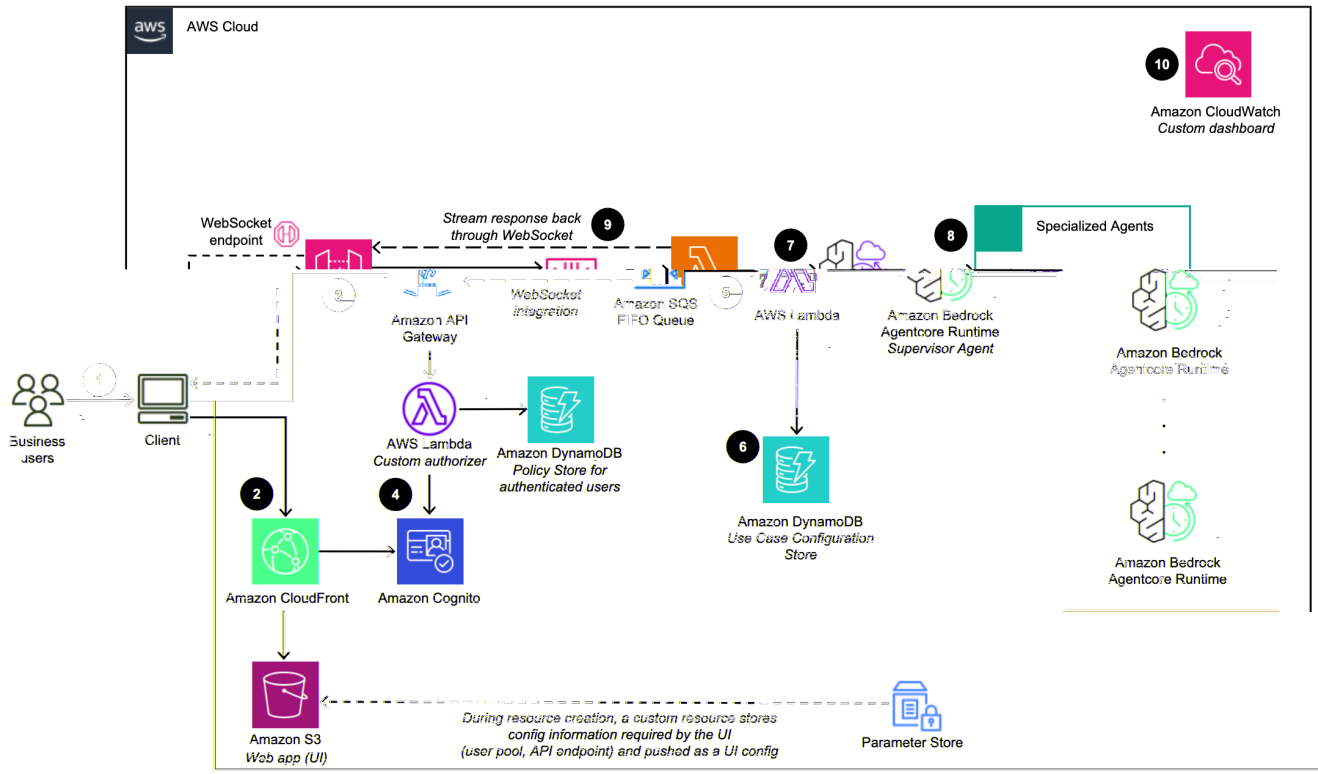
1. Admin users deploy the use case using the Deployment Dashboard. Business users sign in to the use case UI.
2. CloudFront delivers the web UI which is hosted in an S3 bucket.
3. The web UI leverages a WebSocket integration built using API Gateway. The API Gateway is backed by a custom Lambda authorizer function, which returns the appropriate AWS Identity and Access Management(IAM) policy based on the Amazon Cognito group the authenticating user belongs to. The policy is stored in DynamoDB.
4. Amazon Cognito authenticates users and backs both the CloudFront web UI and API Gateway.
5. Incoming requests from the business user are passed from API Gateway to an Amazon SQS queue and then to the AWS Lambda function. The queue enables the asynchronous operation of the API Gateway to Lambda integration. The queue passes connection information to the Lambda function which will then post results directly back to the API Gateway websocket connection to support long running inference calls.
6. The AWS Lambda function retrieves the agent configuration from DynamoDB.

7. Using the user input and any relevant use case configurations, the AWS Lambda function builds and sends a request payload to the agent, running on Amazon Bedrock AgentCore Runtime.
8. The agent connects to associated MCP servers and registers the tools to the strands agent instance. The agent then autonomously selects and performs actions based on tool descriptions and task requirements.
9. When the response comes back from the Amazon Bedrock AgentCore runtime, the Lambda function streams the response back through the API Gateway WebSocket to be consumed by the client application.

> **Note**
>
> - Agent processing is limited to Lambda execution timeout (15 minutes).

Workflow Builder use case

Depicts Workflow Builder architecture



The high-level process flow for the Workflow Builder components deployed with the AWS CloudFormation template is as follows:

1. Admin users deploy the workflow using the Deployment Dashboard, selecting Agent Builder agents to include as specialized agents.
2. CloudFront delivers the web UI which is hosted in an S3 bucket.
3. The web UI leverages a WebSocket integration built using API Gateway. The API Gateway is backed by a custom Lambda authorizer function, which returns the appropriate AWS Identity and Access Management (IAM) policy based on the Amazon Cognito group the authenticating user belongs to. The policy is stored in DynamoDB.
4. Amazon Cognito authenticates users and backs both the CloudFront web UI and API Gateway.
5. Incoming requests from the business user are passed from API Gateway to an Amazon SQS queue and then to the AWS Lambda function. The queue enables the asynchronous operation of the API Gateway to Lambda integration.
6. The AWS Lambda function retrieves workflow configuration from DynamoDB, including the list of specialized Agent Builder agents.
7. Using the user input and workflow configuration, Lambda sends requests to the Amazon Bedrock AgentCore Runtime hosting the supervisor agent.
8. The supervisor agent creates local instances of all specialized Agent Builder agents within the AgentCore Runtime environment. These specialized agents are registered as tools using the Agents as Tools pattern. The supervisor then autonomously selects and delegates work to specialized agents based on agent descriptions and task requirements.
9. The supervisor agent aggregates results from specialized agents and formulates the final response, returning it to the Lambda to be streamed back to the client application through the API Gateway Websocket.

Note

- Workflow processing is limited to Lambda execution timeout (15 minutes).

AWS Well-Architected design considerations

This solution was designed with best practices from the AWS Well-Architected Framework which helps customers design and operate reliable, secure, efficient, and cost-effective workloads in the cloud.

This section describes how the design principles and best practices of the Well-Architected Framework were applied when building this solution.

Operational excellence

This section describes how we architected this solution using the principles and best practices of the operational excellence pillar.

- We built the solution as infrastructure-as-code using Amazon CloudFormation.
- Lambda functions push custom metrics to CloudWatch and a custom CloudWatch dashboard to monitor the health of the solution.
- The solution components are highly modularized, providing the flexibility to choose which components to deploy.

Security

This section describes how we architected this solution using the principles and best practices of the security pillar.

- The Deployment dashboard and all use cases are authenticated and authorized with Amazon Cognito.
- All inter-service communications use AWS IAM roles.
- All solution roles follows least-privilege access; meaning, only the minimum permissions required are granted.
- All data storage including S3 buckets, DynamoDB, and Amazon Kendra have encryption at rest.

Reliability

This section describes how we architected this solution using the principles and best practices of the reliability pillar.

- Architecture based on serverless paradigm.
- We built the architecture for on-demand, horizontal scalability, and automatic recovery from failure of underlying infrastructure.
- The architecture includes buffering and throttling requests to not overwhelm underlying endpoints.

Performance efficiency

This section describes how we architected this solution using the principles and best practices of the performance efficiency pillar.

- The solution uses DynamoDB, a fully managed serverless NoSQL database with on-demand scaling.
- The solution uses Amazon S3 for object storage and to host a website (through CloudFront) to provide low cost, scalable, with 11 9s durability.

Cost optimization

This section describes how we architected this solution using the principles and best practices of the cost optimization pillar.

- Where possible, we built the solution to use serverless architecture; so you only pay for what you use.

Sustainability

This section describes how we architected this solution using the principles and best practices of the sustainability pillar.

- The solution's modular, componentized architecture provides the flexibility to customize resources to be provisioned for individual use cases.
- The architecture uses serverless compute and storage, which optimizes resource utilization.
- As a cloud-based solution, this solution benefits from shared resources, networking, power cooling, and physical facilities.

Architecture details

This section describes the components and AWS services that make up this solution and the architecture details on how these components work together.

AWS services in this solution

AWS service	Description
Amazon API Gateway	**Core**. This service provides the REST APIs for the Deployment dashboard and the WebSocket API for the use case.
AWS CloudFormation	**Core**. This solution is distributed as a CloudFormation template, and CloudForm ation deploys the AWS resources for the solution.
Amazon CloudFront	**Core**. CloudFront serves the web content hosted in Amazon S3.
Amazon Cognito	**Core**. This service handles user management and authentication for the API.
Amazon DynamoDB	**Core**. DynamoDB stores deployment information and configuration details for the Deployment dashboard. It stores chat history and conversation IDs in the Text use case to enable conversation history and query disambiguation.
AWS Lambda	**Core**. The solution uses Lambda functions to: * Back the REST and WebSocket API endpoints * Handle the core logic of each use case orchestrator * Implement custom resources during CloudFormation deployment

AWS service	Description
Amazon S3	**Core**. Amazon S3 hosts the static web content.
Amazon CloudWatch	**Supporting**. This solution publishes logs from solution resources to CloudWatch Logs, and publishes metrics to CloudWatch metrics. The solution also creates a CloudWatch dashboard to view this data.
AWS Systems Manager	**Supporting**. Systems Manager provides application-level resource monitoring and visualization of resource operations and cost data. Also used to store configuration data in Parameter Store.
AWS WAF	**Supporting.** AWS WAF is deployed in front of the API Gateway deployment to protect it.
Amazon Bedrock	**Optional**. The solution leverages Amazon Bedrock to access foundation or customize d models, Amazon Bedrock Agents, Amazon Bedrock Knowledge Bases. Amazon Bedrock is the recommended integration to keep your data from leaving the AWS network.
Amazon Bedrock AgentCore	**Optional** The solution leveraages Amazon Bedrock AgentCore to run and support MCP Server connections as well as Agent Builder and Workflow Use Cases.
Amazon Elastic Container Registry (Amazon ECR)	**Optional**. For Agent Builder deployments, ECR stores and distributes agent container images. The solution uses ECR Pull-Through Cache to automatically retrieve pre-built agent images from the GAAB team's public ECR repository.

AWS service	Description
AWS Distro for OpenTelemetry (ADOT)	**Optional**. For Agent Builder deployments, ADOT provides automatic instrumentation for agent observability, enabling distribut ed tracing and structured logging for agent operations.
Amazon Kendra	**Optional**. In the Text use case, admin users can optionally decide to connect an Amazon Kendra index to use as a knowledge base for the conversation with the LLM. This can be used to inject new information into the LLM giving it the ability to use that information in its responses.
Amazon SageMaker AI	**Optional**. The solution can integrate with an Amazon SageMaker AI inference endpoint to access FMs that are hosted within your AWS account and Region and is a preferred integration to keep your data from leaving the AWS network. **Note** You must deploy the solution in the same Region where the inference endpoint is available.

AWS service	Description
Amazon Virtual Private Cloud	**Optional**. The solution provides the option to deploy components with a VPC-enabled configuration. While deploying the solution with a VPC-enabled configuration, you have the option to let the solution create a VPC for you, or use an existing VPC that exists in the same account and Region where the solution will be deployed (Bring Your Own VPC). If the solution creates the VPC, it creates the necessary network components that includes, subnets, security groups and its rules, route tables, network ACLs, NAT Gateways, Internet Gateways, VPC endpoints, and its policies.

Deployment dashboard

API Gateway custom authorizers

Beneath the surface, Lambda custom authorizers for API Gateway are used for all API calls (both RESTful and WebSocket based) to validate if a given user has permission to perform an action based on the group(s) they belong to. This custom authorizer is backed by a DynamoDB table containing the policies for each group. On invocation of an API, API Gateway invokes the custom authorizer Lambda function, which decodes the provided Amazon Cognito access token to determine which user groups the user belongs to. The policy table is then queried by group name to return the relevant policy for that group.

On every new use case deployment, the admin policy is updated to store a new statement allowing the **execute-api:Invoke** action on that use case's API. When use cases are deleted, the corresponding statement is removed from the policy.

For the groups created for an individual use case, only a single statement is present in the policy, allowing the **execute-api:Invoke** action on only that use case's API.

Due to this structure, any user belonging to a use case's group can access that use case's API. A single user can also be manually added to multiple groups to allow that user to use multiple use cases.

Warning

You can also manually edit the policies for a given group in the policy table if you want to grant access to a new use case to an existing group of users. The use case group is deleted when the use case is deleted (even if you have made manual edits), so proceed with caution when deleting a use case.

In the case where a use case stack is deployed standalone (without the use of the Deployment dashboard), an Amazon Cognito user pool is created for that deployment containing a single user with access to that use case's API. This user pool belongs only to this use case and is not shared across other standalone deployments.

Text use case

Streaming support

In a chat application, latency is an important metric to enable a responsive user experience. The potential for LLM inferences to take from seconds to minutes, provides challenges in how to best serve content to customers. For this reason, several LLM providers allow streaming responses back to the caller. Instead of waiting for the entire inference to complete before returning a response, each token can be returned when it is available.

To support the use of this feature, the Text use case has been designed to use a WebSocket API to back the chat experience. This WebSocket is deployed through API Gateway. The use of a WebSocket API enables a connection to be created at the beginning of a chat session and for responses to be streamed through that socket. This allows frontend applications to provide a better user experience.

Note

Even if a model provides streaming support, this does not necessarily mean that the solution will be able to stream responses back through the WebSocket API. There is

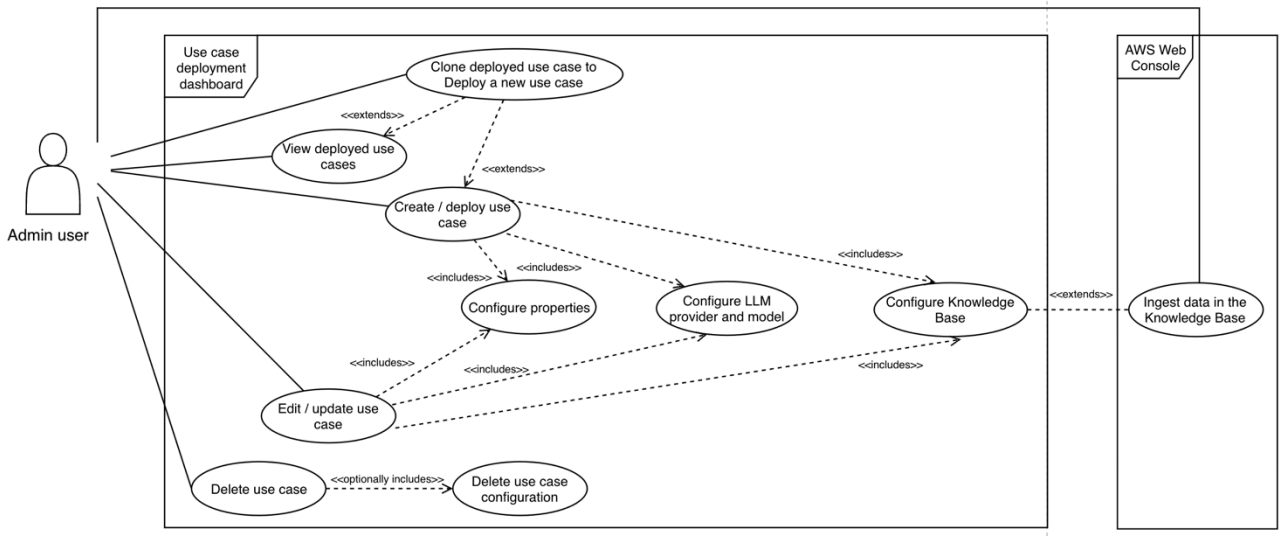
a need for the solution to enable custom logic to support streaming for each model provider. If streaming is available, admin users will be able to enable/disable this feature at deployment time.

How the Generative AI Application Builder on AWS solution works

The admin user primarily interfaces with the Deployment dashboard to view, create, and manage new and existing use case deployments. Through this dashboard, the admin user has access to the following actions:

- View list of deployments
- Create new deployments
- Edit existing deployments
- Clone a deployment's configuration to create a new deployment
- Delete a deployment (deprovision the resources through a CloudFormation delete)
- Permanently delete the configuration details of a deployment

Depicts Use case diagram for the admin user of the Deployment dashboard

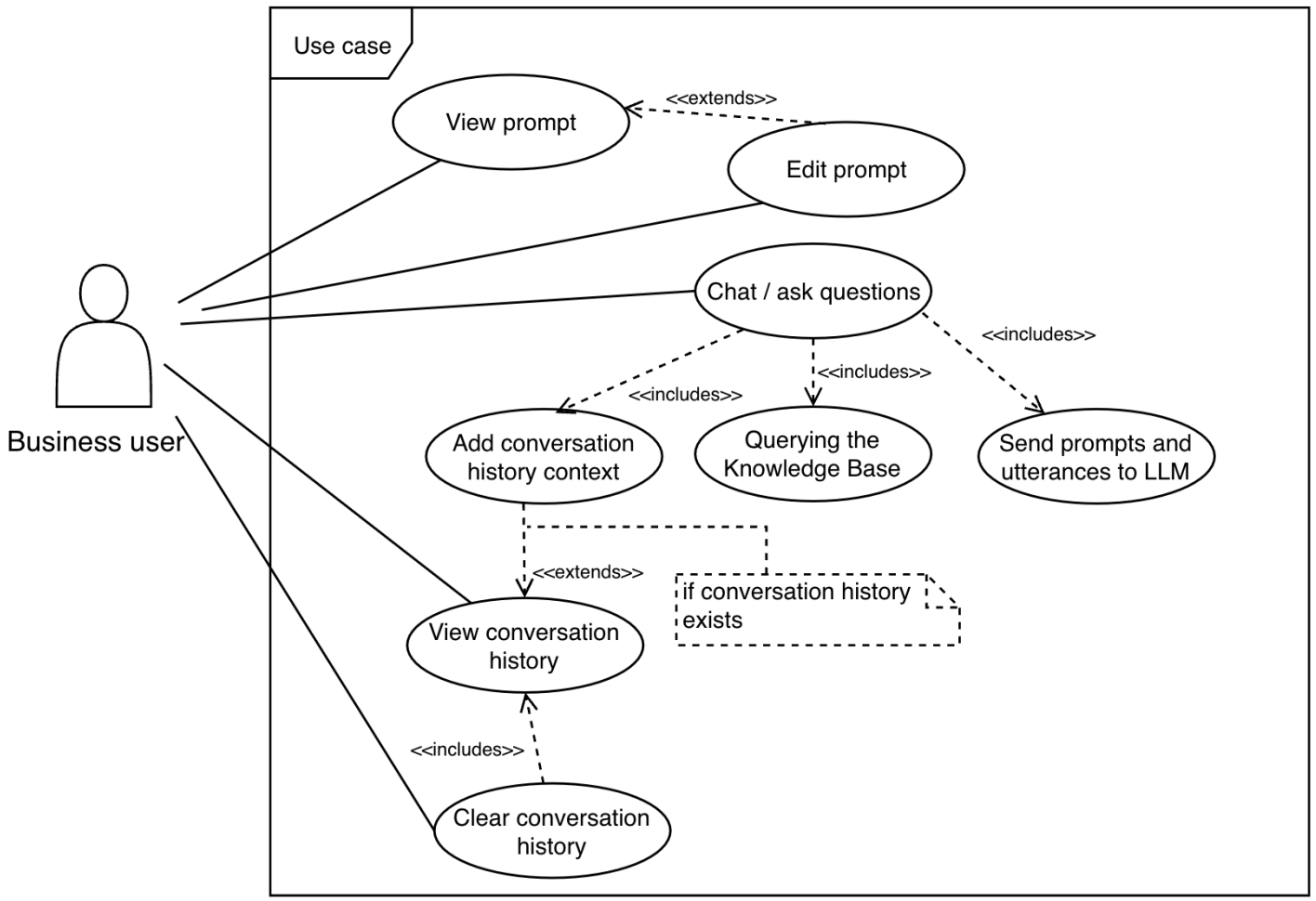


> **Note**
>
> The admin user might not have direct access to the AWS console. In that case, the admin user must work with the DevOps user to support actions such as ingesting data into a Kendra knowledge base.

For the Text use case, the business user gets access to a user interface enabling them to chat with the LLM. The specifics of this configuration are controlled by the deployment settings configured by the admin user. In the Text use case, the business user has access to the following actions:

- Send messages through the chat interface
- View conversation history
- Clear the conversation history
- View prompt
- Edit prompt

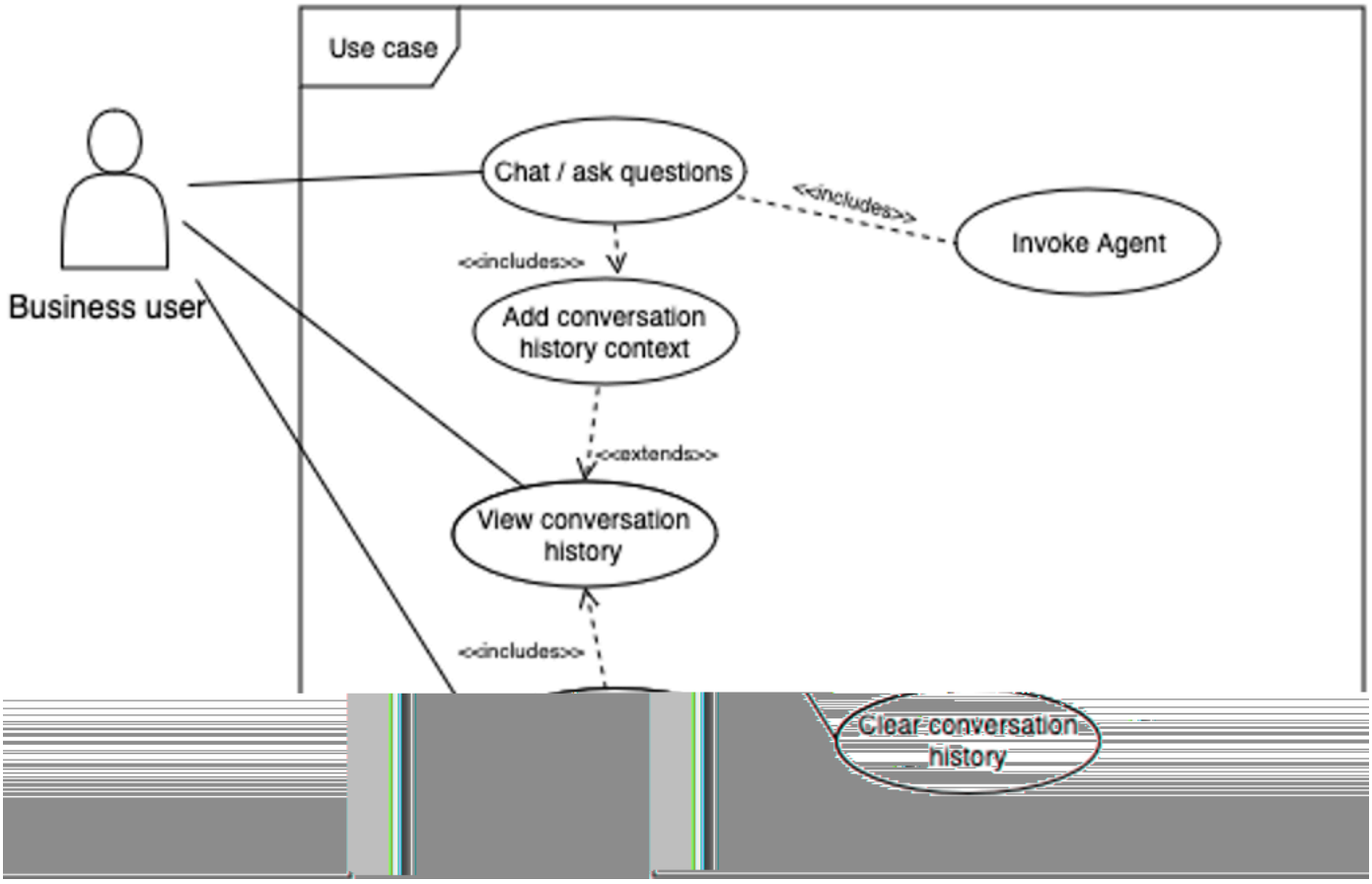
Depicts Use case diagram for the business user of the Text use case



With the Bedrock Agent use case, the business user can access a UI for chatting with the configured Amazon Bedrock Agent. The admin user can configure these specifics in the deployment settings. In the Bedrock Agent use case, the business user has access to the following actions:

- Send messages through the chat interface
- View conversation history
- Clear the conversation history

Depicts Use case diagram for the business user of the Bedrock Agent use case



Agent Builder

The Agent Builder provides a platform for creating, deploying, and managing production-ready AI agents on Amazon Bedrock AgentCore. This section describes the technical components and implementation details.

AgentCore integration

Agent Builder uses a configuration-based deployment approach with pre-built agent images to enable fast, secure, and scalable agent deployments.

Pre-built agent images

Agent container images are built by the GAAB team during the CI/CD pipeline and published to a public ECR repository. Each image version is tied to the GAAB solution version (for example, v4.0.0 → gaab-strands-agent:v4.0.0). Images are based on the Strands SDK and include:

- Agent runtime environment
- MCP client integration
- Memory management capabilities
- OpenTelemetry instrumentation

ECR Pull-Through Cache

The solution uses ECR Pull-Through Cache to automatically distribute agent images from the public ECR repository to the customer's private ECR. This AWS-managed service:

- Caches images on first pull (2-5 minute delay)
- Eliminates custom image copying logic
- Provides local image availability for subsequent deployments
- Creates unique cache rules per deployment to avoid conflicts

Configuration storage

Agent configurations are stored in DynamoDB alongside existing use case configurations. Each configuration includes:

- System prompt template
- Model provider and model ID
- Model parameters (temperature, max_tokens)
- MCP server references and endpoints
- Memory settings (long-term memory toggle)
- Deployment metadata

Image version registry

A DynamoDB table tracks available agent image versions and their cache URIs, enabling version management and backward compatibility.

Agent configuration

System prompts

System prompts define agent behavior, personality, and capabilities. Admin users can:

- Edit the default template through the Agent Builder UI
- Include instructions for tool usage and response formatting
- Reset to default template at any time

Model selection

Agent Builder supports Amazon Bedrock models in v4.0.0:

- Model provider: Amazon Bedrock (only option in v4.0.0)
- Model selection: Claude, Nova, and other Bedrock models
- Model parameters: Temperature, max_tokens, top_p, and model-specific settings

MCP server integration

Model Context Protocol servers provide agents with access to enterprise tools and data:

- Server discovery through GET /mcp API endpoint
- Dynamic configuration without code changes
- Authentication and endpoint management
- Tool capability exposure to agents

Streaming and processing

Real-time streaming

Agent Builder uses Server-Sent Events (SSE) from AgentCore bridged to WebSocket for real-time response streaming:

- Lambda function establishes SSE connection to AgentCore Runtime
- Streams are bridged to API Gateway WebSocket
- Enables token-by-token response delivery to clients

- Maintains connection for long-running requests

Processing constraints

Agent processing in v4.0.0 is limited to Lambda execution timeout:

- Maximum processing time: 15 minutes
- Synchronous processing model
- Suitable for conversational agents and moderate workflows
- Extended async support planned for v4.1+

Memory management

Short-term memory

Enabled by default for all agents using a custom MemoryHookProvider:

- Captures conversation events through Strands callback handlers
- Organizes by actorId and sessionId for context isolation
- Maintains conversation context within sessions
- Automatic integration with AgentCore Memory

Long-term memory

Optional feature using AgentCore Memory Tool from strands_tools:

- Simple toggle in Agent Builder UI
- Semantic memory strategy with default settings
- Agent-controlled access through natural tool invocation
- Stores extracted insights across sessions
- Uses conversationId as sessionId

Observability

AWS OpenTelemetry Distro (ADOT)

Agents are automatically instrumented during container build:

- Automatic trace generation for agent operations
- Distributed tracing across service boundaries
- Structured logging with correlation IDs
- Integration with CloudWatch Transaction Search

Authentication flow

Users authenticate through Amazon Cognito with JWT tokens validated by custom Lambda authorizers that retrieve IAM policies from DynamoDB based on user groups.

Workflow Builder

Workflow Builder enables multi-agent orchestration by creating a supervisor agent that coordinates multiple Agent Builder agents using the Agents as Tools delegation pattern.

Workflow architecture

Key components

- **Supervisor Agent**: Entrypoint agent that receives user requests and delegates to specialized agents
- **Specialized Agents**: Agent Builder use cases registered as tools for the supervisor
- **Agent Registry**: DynamoDB table storing agent configurations and metadata
- **Orchestration Layer**: Strands SDK implementation of Agents as Tools pattern

Agent instantiation

Local agent creation

All specialized agents are instantiated locally within the same AgentCore Runtime:

1. Retrieves agent configurations from DynamoDB
2. Creates local instances of each Agent Builder agent
3. Each agent maintains its own MCP server connections
4. Supervisor agent registers specialized agents as tools

5. Strands SDK manages agent selection and delegation

Plan your deployment

This section describes the cost, security, Region, and quota considerations for planning your deployment.

> **Important**
>
> This solution leverages Amazon Bedrock as the primary service for accessing AI-generated models. You must first request access to models before they are available for use within the solution. For details, refer to Model access in the *Amazon Bedrock User Guide*.

Supported AWS Regions

> **Important**
>
> This solution optionally uses the Amazon Bedrock and Amazon Kendra services, which are not currently available in all AWS Regions. You must launch this solution in an AWS Region where these services are available. For the most current availability of AWS services by Region, see the AWS Regional Services List.

Generative AI Application Builder on AWS is supported in the following AWS Regions:

Region name	
US East (Ohio)	Canada (Central)
US East (N. Virginia)	Europe (Frankfurt)
US West (Northern California)	Europe (Ireland)
US West (Oregon)	Europe (London)
Asia Pacific (Mumbai)	Europe (Milan)
Asia Pacific (Seoul)	Europe (Paris)

Region name	
Asia Pacific (Singapore)	Europe (Stockholm)
Asia Pacific (Sydney)	Middle East (Bahrain)
Asia Pacific (Tokyo)	South America (São Paulo)

Note

If using a foundation model accessed outside of AWS in your deployments, check with the model provider which Regions their APIs are available in. If their APIs are only available in certain Regions, you might experience instability in the form of high latency or even time outs. It's also important to check with your organization's legal and compliance teams to evaluate the considerations of data crossing regional boundaries.

Cost

With this AWS Solution, you pay only for the resources you use and there are no minimum fees or setup charges. Users pay for the dashboard used to launch Generative AI use cases and, and for any use cases that are deployed. The cost of deployed use cases depends on the configurations. Example configurations:

1. A simple Deployment dashboard which costs approximately $20 USD per month.
2. A simple production-ready chatbot use case deployed with default settings running in US East (N. Virginia), powered by Amazon Bedrock without access to documents, which also costs around $200 USD per month.
3. A scaled system in an Amazon VPC use case that supports 8,000 queries per day over tens of thousands of documents, which costs around $1,500 USD per month. The cost of the use case will vary depending on the configuration, such as Text use cases with different model providers, with or without Retrieval Augmented Generation (RAG) enabled, and so on.

Workload description	Estimated cost (USD/month)
Sample cost for Deployment dashboard	$20/month
Sample costs for a text-based proof of concept (includes Deployment dashboard and 1 Text use case, ~100 interactions per day)	$40/month
Sample costs for a highly scalable generative AI query engine (Includes Deployment dashboard, 1 Text use case, and an Amazon Kendra Index for RAG up to 100K documents with ~8K queries per day, with VPC enabled	$1,500/month
Sample costs for an agent-based proof of concept (Includes Deployment dashboard, 1 Bedrock Agent use case with Amazon Bedrock Knowledge Bases and Amazon Bedrock Guardrails enabled, ~100 interactions per day)	$840/month
Sample costs for MCP Server (Includes Deployment dashboard, 1 MCP Server use case with Gateway method for Lambda integration, ~100 tool invocations per day)	$22/month
Sample costs for Agent Builder (Includes Deployment dashboard, 1 Agent Builder use case with MCP integration and long-term memory enabled, ~100 interactions per day)	$55/month
Sample costs for Workflow Builder	$109/month

Workload description	Estimated cost (USD/month)
(Includes Deployment dashboard, 1 Workflow with 3 Agent Builder agents, ~100 interactions per day)	

Important

These examples are only intended to help you estimate the costs for your specific workloads. The use of different LLMs, configurations, or AWS services can change your costs (example, serverless/on-demand billing vs. provisioned/time-billed). To manage costs, we recommend creating a budget through AWS Cost Explorer. Prices are subject to change. For full details, refer to the pricing webpage for each AWS service used in this solution.

Sample costs for running the Deployment dashboard

The following table provides the cost breakdown for a Deployment dashboard with default parameters and 100 active users in the US East (N. Virginia) Region for one month, which will cost about $20/month.

AWS service	Dimensions	Cost [USD]
API Gateway, DynamoDB, CloudFront, Amazon S3, Lambda, Systems Manager Parameter Store	5,000 512 KB REST API calls per month without caching enabled	$1.97
Amazon Cognito	100 active users per month with advanced security features enabled and no users signing in through SAML or OIDC federation	$5.55

AWS service	Dimensions	Cost [USD]
AWS WAF	10,000 web requests across 1 web ACL and 7 defined rules without any rule groups	$12.60
Total Deployment dashboard cost		**$20.12**

Sample costs for a text-based proof of concept

A Deployment dashboard can have many use cases deployed at a given time. The following table shows the cost breakdown of a use case deployed without RAG for 1 business user performing 100 queries per day with the LLM. Queries are sent as a text message on the WebSocket and the response is streamed back as tokens with the assumption that streaming is enabled. Using the Amazon Bedrock Nova Pro model, the cost of running this use case is about $20/month.

AWS service	Dimensions	Cost [USD]
API Gateway (WebSocket), CloudFront, Lambda, Amazon S3, AWS Systems Manager Parameter Store	100 chat interactions per day. Average message size 32 KB per message and 5 minutes per connection.	$0.61
CloudWatch	1.5 GB CloudWatch logs with verbose mode on for experimentation	$7.23
Amazon DynamoDB	Conversation history table, 1 GB storage LLM configuration table, 1 GB storage	$3.05
Subtotal of the use case costs (not including LLMs)		**$10.89**

AWS service	Dimensions	Cost [USD]
Amazon Bedrock (Nova Pro)	Assumptions for 100 interactions per day: * Monthly cost for 190K input tokens per day = $0.152 × 30 * Monthly cost for 16K output tokens per day = $0.0512 × 30	$6.10
Total application cost with Amazon Bedrock (Nova Pro)	**$10.89 (Use Case cost) + $6.10 (Amazon Bedrock cost)**	**$17.00**

Note

The costs of inference calls made to services outside the AWS network are not included in these estimates. Refer to the pricing guide of your LLM provider if you're not using an AWS model provider.
Pricing guides for AWS services can be found at: Amazon Bedrock pricing and Amazon SageMaker AI pricing.

Sample costs for a highly scalable generative AI query engine

The following table provides the cost breakdown of a RAG-enabled use case with Amazon Bedrock's Nova Pro model as the LLM. When a Bedrock Knowledge Base is added, this use case costs about $1300/month

AWS service	Dimensions	Cost [USD]
API Gateway (WebSocket)	8000 chat interactions per day. Average message size 32 KB per message and 5 minutes per connection.	$38.89

AWS service	Dimensions	Cost [USD]
CloudFront	240,000 requests per month with 100 GB data transferred out to the internet and 1 GB data transferred out to the origin	$8.76
Amazon Bedrock (Nova Pro)	Assumptions: Input tokens = promptTemplate (400) + context (400)+ chatHistory (1080) + query Input tokens (20)= 1,900 Output tokens = 160 (average) With 8,000 transactions a day, Daily Input Tokens cost (1,900 x 8,000 = 15,200,000 tokens x 0.0008/1000 price per token) Daily Output Tokens cost (160 x 8,000 = 1,280,000 tokens x 0.0032/1000 price per token) Monthly cost (($12.16 + $4.10) x 30)	$487.80
CloudWatch	24 metrics using 5 GB data ingested for logs and 1 dashboard	$9.72

AWS service	Dimensions	Cost [USD]
DynamoDB	DynamoDB table to keep track of conversation history with each record up to 1 KB data, 8,000 read and writes per day	$11.70
Lambda	Container size - 128 MB, 512 MB ephemeral storage, 2 Lambda functions used for authorization Container size - 256 MB, 512 MB ephemeral storage, 5 requests per second with 20 seconds average compute time	$20.89
Total use case cost		**$577.76/month + knowledge base cost (see below)**

> **Note**
>
> The costs of API calls made to any services outside of the AWS network are not included in these estimates. See the pricing guide of your LLM provider if not using Amazon Bedrock.

Costs for adding a knowledge base

Knowledge base costs will vary based on the type of knowledge base used, and (in the case of Bedrock) the backing vector store used by the knowledge base. Provisioning and managing the knowledge bases is outside of the scope of the solution.

Amazon Bedrock Knowledge Bases

The solution does not manage or provision any resources related to Amazon Bedrock Knowledge Bases. Amazon Bedrock does not incur cost for using the knowledge base feature itself, however you will be charged for the usage of the embedding model used by your use case on each query. Additionally, the backing vector store for your knowledge base (for example, an index in Amazon OpenSearch Service, or a database inside Amazon Relational Database Service) will have an associated cost which cannot be provided or calculated here.

For the above highly scalable generative AI query engine scenario, the costs incurred by this service for calling the Amazon Bedrock embeddings model are as follows:

AWS service	Dimensions	Cost [USD]
Amazon Bedrock (Amazon Titan Text Embeddings V2)	8,000 queries a day with 1,900 input tokens per query = 15,200,000 tokens = $0.30 USD per day. Daily cost x 30 days = $9.00 USD monthly cost	$9.00
Amazon OpenSearch Service (Serverless) Sample Usage	Basic serverless configura tion with 4 x OpenSearch Compute Unit (OCU) (billable minimum) = $23.04 USD per day Daily cost x 30 days = $691.20 USD **Note** This provides a rough estimate, as some workloads will require more OCUs, while customers with existing provisioned OpenSearch resources	$691.20

AWS service	Dimensions	Cost [USD]
	will incur less cost here.	
Total additional cost		$ 700.20

Amazon Kendra

The solution can provision a Kendra index for you, or you can bring your own. The cost for running a configuration suited to the above highly scalable generative AI query engine is as follows:

AWS service	Dimensions	Cost [USD]
Amazon Kendra	0-8,000 queries a day and up to 100,000 documents with Amazon Kendra Enterprise Edition with 0-50 data sources	$1,008.00

Note

You can share the Amazon Kendra index between use cases, but this can drive up the number of queries per index. If this falls outside the Amazon Kendra Enterprise edition, additional charges will apply.

Incremental cost of enabling Amazon VPC for a use case

The following table provides the cost breakdown of enabling Amazon VPC for a use case deployed in two AZs.

AWS service	Dimensions	Cost [USD]
Amazon NAT Gateway	Assumption: 2 AZ deployment, with a NAT Gateway in each	$74.70

AWS service	Dimensions	Cost [USD]
	AZ. 100 GB of data processed through NAT Gateway 730 hours, 100 GB data processed per month	
AWS PrivateLink (VPC Endpoints)	Assumptions: 2 AZ deployment, with 1 private subnet in each AZ and 1 VPC Endpoint with 2 elastic network interfaces (ENIs). 6 VPC endpoints, 2 ENIs per VPC endpoint, 730 hours with 1,024 GB data processed in a month	$97.84
Public IPv4 address	Assumption: 2 AZ deployment, 1 public subnet in each AZ with a NAT Gateway in each public subnet. Each NAT Gateway configured with 1 active public IPv4. 2 active public IPv4 address x 730 hours in a month x $0.005 hourly charge = $7.3 USD	$7.30
Additional cost (for Amazon VPC)		**$179.93**

Cost implications when using Provisioned Throughput

Provisioned throughput costs will vary based on the type of model you've provisioned and your commitment period as well as Model Units selected for the commitment period. There is an additional cost associated with using Provisioned Throughput.

For more information and the most up-to-date pricing, you can refer to Bedrock Pricing.

Cost for using cross-region inference

There is no additional cost for routing or data transfer for using cross-region inference. You pay the same price per token for models as in your source or primary Region.

Sample costs for an agent-based proof of concept

When you use Amazon Bedrock Agents, you're charged based on the components comprising the agent, such as the backing model and knowledge base (if RAG is enabled), along with additional capabilities that you add. The following table shows the cost breakdown of a Bedrock Agent use case configured with an on-demand Claude 3.5 Sonnet model, Amazon Bedrock Knowledge Bases, and Amazon Bedrock Guardrails.

Similar to the cost for adding Amazon Bedrock Knowledge Bases, this solution doesn't manage or provision resources related to Amazon Bedrock Agents. The solution also doesn't incur cost for using Amazon Bedrock Knowledge Bases, but does incur cost for:

- Using the embedding model for each query that is sent to it
- The backing vector store for your knowledge base (for example, an index in Amazon OpenSearch Service, or a database inside Amazon RDS)

The following table assumes 100 interactions per day with 1,900 input tokens and 160 output tokens per query.

> **Note**
>
> For this sample Bedrock Agent use case, if there were an action group configured to use an external API, those costs would be additional. They are outside the scope of the calculations in this table.

AWS service	Dimensions	Cost [USD]
API Gateway (WebSocket), CloudFront, Lambda, Amazon S3, Systems Manager Parameter Store	100 chat interactions per day, average message size 32 KB per message, 5 minutes per connection	$0.61
CloudWatch	1.5 GB CloudWatch Logs with verbose mode on for experimentation	$7.23
DynamoDB	LLM configuration table for 1KB record size and 1 GB storage	$0.25
Subtotal of costs (not including LLMs)		**$8.09**
Anthropic Claude 3.5 Sonnet	* Daily cost for 190K input tokens per day (0.003/1,000 tokens) = $0.57 + Daily cost × 30 days = $17.10 * Daily cost for 16K output tokens per day (0.015/1,000 tokens) = $0.24 + Daily cost × 30 days = $7.20	$24.30
Amazon Bedrock (Amazon Titan Text Embeddings V2) for Amazon Bedrock Knowledge Bases	Daily cost for 190K input tokens per day (0.00002/1000 tokens) = 0.004 Daily cost × 30 days = $0.12	$0.12
Amazon OpenSearch Service (Serverless) sample usage	Basic serverless configuration with 4 × OpenSearch Compute Unit (OCU) (billable minimum) = $23.04 per day	$691.20

AWS service	Dimensions	Cost [USD]
	Daily cost × 30 days = $691.20	
Amazon Bedrock Guardrails	190K tokens is roughly equivalent of 760K (190,000 × 4) characters and 3,800 text units (760K characters / 200) Consider a guardrail configured with content filters, personally identifiable information (PII) filter, sensitive information filter (regular expression) and word filters Daily content filter cost (0.75/1000 text units) + PII filter cost ($0.1/1000 text units) + sensitive information filter (regex) + word filters = $2.85 + $0.38 + $0 + $0 Monthly cost = Daily cost × 30 days = $96.90	$96.90
Total application cost for an agent backed by Anthropic Claude 3.5 Sonnet	*$8.09 (use case cost) +* **$812.52 (other agent configurations)**	$820.61

Note

Refer to the pricing guide of your LLM provider if you're not using an AWS model provider. Pricing guides for AWS services can be found at: Amazon Bedrock pricing and Amazon SageMaker AI pricing.

Sample costs for MCP Server

MCP Server use cases enable deployment and management of Model Context Protocol servers on Amazon Bedrock AgentCore. The following table shows the cost breakdown of an MCP Server use case using the Gateway method to wrap existing Lambda functions.

The solution manages the AgentCore Gateway deployment and configuration. You're charged for:

- Infrastructure costs (API Gateway, Lambda, DynamoDB, CloudWatch, S3)
- AgentCore Gateway consumption (per tool invocation)
- Lambda function execution costs (for Gateway method with Lambda targets)
- External API costs (for Gateway method with API or MCP Server targets, if applicable)

Item	Calculations	Cost
Amazon API Gateway (REST API)	100 tool invocations per day × 30 days = 3,000 requests per month	$0.05
AWS Lambda (orchestration)	100 invocations per day × 30 days × 1 second average × 512 MB = 3,000 GB-seconds per month	$0.05
Amazon DynamoDB	3,000 read/write requests per month + 1 GB storage	$0.15
Amazon CloudWatch	Standard monitoring and logging for 3,000 invocations	$1.00
Amazon S3	Configuration storage and logs (minimal usage)	$0.25
Amazon Bedrock AgentCore Gateway	3,000 tool invocations per month	$0.05
Target Lambda Function	100 invocations per day × 30 days × 0.5 seconds × 128	$0.25

Item	Calculations	Cost
	MB = 1,500 GB-seconds per month	
Total monthly cost	*$1.75 (infrastructure) + $0.05 (AgentCore Gateway)*	$1.80

Note

Costs vary based on deployment method (Gateway vs Runtime), target types, and usage patterns. Runtime method deployments incur AgentCore Runtime charges instead of Gateway charges. External API costs and custom container hosting costs are additional.

Sample costs for Agent Builder

Agent Builder enables you to create and deploy custom agents on Amazon Bedrock AgentCore. The following table shows the cost breakdown of an Agent Builder use case configured with Claude 3.5 Sonnet, MCP server integration, and long-term memory enabled.

The solution manages the AgentCore Runtime deployment and configuration. You're charged for:

- Infrastructure costs (API Gateway, Lambda, DynamoDB, CloudWatch, S3)
- AgentCore Runtime consumption (CPU and memory hours based on actual agent execution time)
- Foundation model inference (input and output tokens)
- AgentCore Memory (short-term events and long-term storage/retrieval)

The following table assumes 100 interactions per day with 1,900 input tokens and 160 output tokens per query, with an average agent execution time of 5 seconds per interaction.

AWS service	Dimensions	Cost [USD]
API Gateway (WebSocket), CloudFront, Lambda,	100 chat interactions per day, average message size 32 KB	$0.61

AWS service	Dimensions	Cost [USD]
Amazon S3, Systems Manager Parameter Store	per message, 5 minutes per connection	
CloudWatch	1.5 GB CloudWatch Logs with verbose mode on for experimentation	$7.23
DynamoDB	LLM configuration table for 1KB record size and 1 GB storage	$0.25
Subtotal of infrastructure costs		**$8.09**
Amazon Bedrock AgentCore Runtime	* CPU: 1 vCPU × 5 seconds × 100 interactions = 125 vCPU-seconds/day = 0.140 vCPU-hours/day + Daily cost: 0.140 × $0.0895 = $0.013 + Monthly cost: $0.013 × 30 = $0.38 * Memory: 512 MB (0.5 GB) × 5 seconds × 100 interactions = 250 GB-seconds/day = 0.069 GB-hours/day + Daily cost: 0.069 × $0.00945 = $0.0007 + Monthly cost: $0.0007 × 30 = $0.02	$0.40

AWS service	Dimensions	Cost [USD]
Anthropic Claude 3.5 Sonnet	* Daily cost for 190K input tokens per day (0.003/1,000 tokens) = $0.57 + Daily cost × 30 days = $17.10 * Daily cost for 16K output tokens per day (0.015/1,000 tokens) = $0.24 + Daily cost × 30 days = $7.20	$24.30
Amazon Bedrock AgentCore Memory	* Short-term memory: 100 new events/day × $0.25/1,000 events = $0.025/day + Monthly cost: $0.025 × 30 = $0.75 * Long-term memory storage (built-in strategy): 100 records × $0.75/1,000 records/month = $0.075/month * Long-term memory retrieval: 100 retrievals/day × $0.50/1,000 retrievals = $0.05/day + Monthly cost: $0.05 × 30 = $1.50	$2.33
Total application cost for Agent Builder with Claude 3.5 Sonnet	*$8.09 (infrastructure) + $0.40 (AgentCore Runtime) + $24.30 (model) + $2.33 (memory)*	**$35.12**

Note

AgentCore Runtime pricing is consumption-based. Actual costs depend on:

- Agent execution time (CPU and memory usage during active processing)
- Number of interactions and their complexity
- MCP tool usage (additional CPU/memory for tool execution)
- Memory configuration (short-term vs. long-term memory enabled)

For detailed AgentCore pricing, refer to Amazon Bedrock pricing.

Note

If using MCP servers that invoke external APIs or services, those costs are additional and outside the scope of this calculation. Similarly, if using AgentCore Browser or Code Interpreter tools, consumption-based charges apply at $0.0895 per vCPU-hour and $0.00945 per GB-hour.

Sample costs for Workflow Builder

Workflow Builder creates a supervisor agent that orchestrates multiple Agent Builder agents. The following table shows the cost breakdown for a workflow with 1 supervisor agent and 3 specialized Agent Builder agents, all configured with Claude 3.5 Sonnet and long-term memory enabled.

Assumptions: 100 interactions per day, average 2 agent delegations per interaction, 5 seconds execution time per agent.

AWS service	Dimensions	Cost [USD]
API Gateway (WebSocket), CloudFront, Lambda, Amazon S3, Systems Manager Parameter Store	100 chat interactions per day, average message size 32 KB per message, 5 minutes per connection	$0.61
CloudWatch	1.5 GB CloudWatch Logs with verbose mode on for experimentation	$7.23

AWS service	Dimensions	Cost [USD]
DynamoDB	LLM configuration table for 1KB record size and 1 GB storage	$0.25
Subtotal of infrastructure costs		**$8.09**
Amazon Bedrock AgentCore Runtime (Supervisor Agent)	* CPU: 1 vCPU × 5 seconds × 100 interactions = 0.140 vCPU-hours/day × 30 = $0.38 * Memory: 0.5 GB × 5 seconds × 100 interactions = 0.069 GB-hours/day × 30 = $0.02	$0.40
Amazon Bedrock AgentCore Runtime (3 Specialized Agents)	* Average 2 delegations per interaction = 200 agent executions/day * CPU: 1 vCPU × 5 seconds × 200 = 0.278 vCPU-hours/day × 30 = $0.75 * Memory: 0.5 GB × 5 seconds × 200 = 0.139 GB-hours/day × 30 = $0.04	$0.79
Anthropic Claude 3.5 Sonnet (Supervisor Agent)	* Input: 190K tokens/day × $0.003/1K = $0.57/day × 30 = $17.10 * Output: 16K tokens/day × $0.015/1K = $0.24/day × 30 = $7.20	$24.30
Anthropic Claude 3.5 Sonnet (Specialized Agents)	* Average 2 delegations per interaction * Input: 380K tokens/day × $0.003/1K = $1.14/day × 30 = $34.20 * Output: 32K tokens/day × $0.015/1K = $0.48/day × 30 = $14.40	$48.60

AWS service	Dimensions	Cost [USD]
Amazon Bedrock AgentCore Memory (Supervisor Agent)	* Short-term: 100 events/day × $0.25/1K × 30 = $0.75 * Long-term storage: 100 records × $0.75/1K = $0.08 * Long-term retrieval: 100 retrievals/day × $0.50/1K × 30 = $1.50	$2.33
Amazon Bedrock AgentCore Memory (Specialized Agents)	* Short-term: 200 events/day × $0.25/1K × 30 = $1.50 * Long-term storage: 200 records × $0.75/1K = $0.15 * Long-term retrieval: 200 retrievals/day × $0.50/1K × 30 = $3.00	$4.65
Total application cost for Workflow Builder with 3 agents	*$8.09 (infrastructure) + $1.19 (AgentCore Runtime) + $72.90 (models) + $6.98 (memory)*	**$89.16**

Note

- Higher delegation rates increase token consumption proportionally

For detailed AgentCore pricing, refer to Amazon Bedrock pricing.

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared responsibility model reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, virtualization layer, and physical security of the facilities in which the services operate. For more information about AWS security, visit AWS Cloud Security.

Using foundation models on Amazon Bedrock

Amazon Bedrock hosts a collection of models from Amazon Nova models to other leading foundation models (FMs). When using Amazon Bedrock, all models are hosted within the AWS infrastructure. This means that when using Amazon Bedrock as the LLM provider, all of your inference requests will remain within the AWS network and network traffic will not leave your Region.

> **Note**
>
> All foundation models (FMs) available through Amazon Bedrock are hosted directly on AWS infrastructure managed and owned by AWS. Model providers do not have access to customer data such as prompts and continuations, or Amazon Bedrock service logs. For additional information about Amazon Bedrock's security posture, refer to Data protection in Amazon Bedrock in the *Amazon Bedrock User Guide*.

IAM roles

IAM roles allow customers to assign granular access policies and permissions to services and users on the AWS Cloud. This solution creates IAM roles that grant the solution's Lambda functions access to create Regional resources.

CloudWatch Logs

You can enable verbose mode while deploying a use case using the Deployment Dashboard model selection page, under Additional Settings. Verbose mode enables detailed CloudWatch logs which can be helpful for debugging and prompt experimentation.

> **Note**
>
> When verbose mode is enabled, retrieved documents from the knowledge base (if RAG is enabled) and prompts will also be logged, which may contain sensitive information.

VPC

The solution provides two options for Amazon VPC configuration:

1. Let the solution build an Amazon VPC for you.
2. Managing and bringing your own Amazon VPC for use within the solution.

Let the solution build an Amazon VPC for you

If you select the option to let the solution build an Amazon VPC, it will deploy as a 2-AZ architecture by default with a CIDR range 10.10.0.0/20. You have the option to use Amazon VPC IP Address Manager (IPAM), with 1 public subnet and 1 private subnet in each AZ. The solution creates NAT Gateways in each of the public subnets, and configures Lambda functions to create the ENIs in the private subnets. Additionally, this configuration creates route tables and its entries, security groups and its rules, network ACLs, VPC endpoints (gateway and interface endpoints).

Managing your own Amazon VPC

When deploying the solution with an Amazon VPC, you have the option to use an existing Amazon VPC in your AWS account and Region. We recommended that you make your VPC available in at least two availability zones to ensure high availability. Your VPC must also have the following VPC endpoints and their associated IAM policies for your VPC and route table configurations.

For a Deployment dashboard Amazon VPC

1. Gateway endpoint for DynamoDB.
2. Gateway endpoint for S3.
3. Interface endpoint for CloudWatch.
4. Interface endpoint for AWS CloudFormation.

For a use case Amazon VPC

1. Gateway endpoint for DynamoDB.
2. Gateway endpoint for S3.
3. Interface endpoint for CloudWatch.
4. Interface endpoint for Systems Manager Parameter Store.

> **Note**
>
> The solution only requires `com.amazonaws.region.ssm`.

5. Interface endpoint for Amazon Bedrock (bedrock-runtime, agent-runtime, bedrock-agent-runtime).
6. Optional: If the deployment will use Amazon Kendra as a knowledge base, then an interface endpoint for Amazon Kendra is needed.
7. Optional: if the deployment will use any LLM under Amazon Bedrock, then an interface endpoint for Amazon Bedrock is needed.

   Note

   The solution only requires `com.amazonaws.region.bedrock-runtime`.

8. Optional: If the deployment will use Amazon SageMaker AI for the LLM, then an interface endpoint for Amazon SageMaker AI is needed.

Note

The solution will not delete or modify the VPC configuration when using the **Bring your own VPC deployment** option. However, it will delete any VPCs that are created by the solution in the **Create a VPC for me** option. For this reason, you must be careful when sharing a solution-managed VPC across stacks/deployments.
For example, deployment A uses **Create a VPC for me** option. Deployment B uses **Bring my own VPC** using the VPC created by deployment A. If deployment A is deleted before deployment B, then deployment B will no longer work because the VPC has been deleted. Also because deployment B is using the ENIs created by the Lambda functions, deleting deployment A might have errors and retention of residual resources.

Amazon CloudFront

This solution deploys a web console hosted in an Amazon S3 bucket. To help reduce latency and improve security, this solution includes a CloudFront distribution with an origin access identity, which is a CloudFront user that provides public access to the solution's website bucket contents. For more information, see Restricting Access to Amazon S3 Content by Using an Origin Access Identity in the *Amazon CloudFront Developer Guide*.

> **Note**
>
> CloudFront has an account-level soft quota limit of 20 response header policies. This solution creates custom response header policies for security purposes. If you have more than 20 deployments of the Generative AI Application Builder on AWS or its use cases, new deployments may fail due to hitting the quota limit.

To resolve this issue, you can request a quota increase for the **Response Header Policies** quota in the AWS Service Quotas console by following these steps:

1. Open the AWS Service Quotas console.
2. In the navigation pane, select **AWS services**.
3. Search for and select **Amazon CloudFront**.
4. Scroll to the **Response Header Policies** quota and choose **Request quota increase**.
5. Follow the prompts to request an increase in the quota limit for your AWS account.

By increasing the **Response Header Policies** quota, you can ensure that new deployments of the Generative AI Application Builder on AWS or its use cases do not fail due to the quota limit.

Quotas

Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account.

Quotas for AWS services in this solution

Make sure you have sufficient quota for each of the services implemented in this solution. For more information, refer to AWS service quotas.

Use the following links to go to the page for that service. To view the service quotas for all AWS services in the documentation without switching pages, view the information in the Service endpoints and quotas page in the PDF instead.

Amazon Bedrock AgentCore quotas

For Agent Builder deployments, be aware of the following Amazon Bedrock AgentCore service quotas:

Quota	US East (N. Virginia)	Other Regions
Active Session workloads per account	1000	500
Total agents per account	1,000	1,000
Versions per account	1,000	1,000

Deploy the solution

This solution uses AWS CloudFormation templates and stacks to automate its deployment. The CloudFormation template specifies the AWS resources included in this solution and their properties. The CloudFormation stack provisions the resources that are described in the template.

Deployment process overview

Before you launch the solution, review the cost, architecture, security, and other considerations discussed in this guide.

> **Important**
>
> If you plan to use Amazon Bedrock, you must request access to models before they are available for use. Refer to Model access in the *Amazon Bedrock User Guide* for more details.

Time to deploy: Approximately 10 minutes

Step 1: Launch the Deployment dashboard stack

Step 2: Deploy a use case

Step 3: Deploy a use case using the Deployment dashboard wizard

Step 4: Post-deployment configuration

Optionally, you can deploy the use cases separately from the solution, if you prefer not to have the Deployment dashboard UI or APIs.

- Deploying a standalone Text use case
- Deploying a standalone Bedrock Agent use case

You can also supply a DynamoDB chat configuration.

Important

This solution sends operational metrics to AWS (the "Data") about the use of this solution. We use this Data to better understand how customers use this solution and related services and products. AWS's collection of this Data is subject to the AWS Privacy Policy.

AWS CloudFormation template

You can download the CloudFormation template for this solution before deploying it.

View template

generative-ai-application-builder-on-aws.template - Use this template to launch the solution and all associated components. The default configuration deploys the core and supporting solutions found in the AWS services in this solution section, but you can customize the template to meet your specific needs.

Note

AWS CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs.

This AWS CloudFormation template deploys Generative AI Application Builder on AWS in the AWS Cloud.

Step 1: Launch the Deployment dashboard stack

Follow the step-by-step instructions in this section to configure and deploy the solution into your account.

Time to deploy: Approximately 10 minutes

1. Sign in to the AWS Management Console and select the button to launch the `generative-ai-application-builder-on-aws.template` CloudFormation template.

   Launch solution

2. The template launches in the US East (N. Virginia) Region by default. To launch the solution in a different AWS Region, use the Region selector in the console navigation bar.

> **Note**
>
> This solution uses Amazon Kendra and Amazon Bedrock, which are not currently available in all AWS Regions. If using these features, you must launch this solution in an AWS Region where these services are available. For the most current availability by Region, see the AWS Regional Services List.

3. On the **Create stack** page, verify that the correct template URL is in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, assign a name to your solution stack. For information about naming character limitations, see IAM and STS Limits in the *AWS Identity and Access Management User Guide*.
5. Under **Parameters**, review the parameters for this solution template and modify them as necessary. This solution uses the following default values.

Parameter	Default	Description
Admin User Email	No	The email address of the admin user who will have access to the Deployment dashboard. If provided, an Amazon Cognito group and user will be created with permissions to deploy and manage use cases. You may also use `placeholder@example.com` to create the Group but not the User. Refer to Manual User Pool Configuration for information on setting up your user pool.

Parameter	Default	Description
VpcEnabled	No	Should the Deployment dashboard be deployed within a VPC
CreateNewVpc	No	Only available, if **VpcEnabled** is Yes. If the value is Yes, the stack will create the VPC and deploy the solution within the created VPC. If **VpcEnabled** is Yes and **CreateNewVpc** is No, then you must provide an existing VPC configuration (**ExistingVpcId**, **ExistingPrivateSubnetIds**, **ExistingSecurityGroupIds**, **VpcAzs**).
IPAMPoolId	*(Optional input)*	You can configure IPAM and provide the created id as input to assign the IP address range that the deployment of this stack should use. For details regarding IPAM, see How IPAM works.

Parameter	Default	Description
DeployUI	`Yes`	You have the option to deploy the Deployment dashboard without the web user interface (and the AWS resources required for the web deployment). In which case, the solution will deploy all infrastructure including REST API endpoints. This option is useful to integrate your own web interface with the Deployment dashboard APIs.
ExistingVpcId	*(Optional input)*	Required only if you want to deploy the solution in an existing VPC that you have created.
ExistingPrivateSubnetIds	*(Optional input)*	Required only if you want to deploy the solution in an existing VPC that you have created. The Lambda functions will be deployed in this subnet.
ExistingSecurityGroupIds	*(Optional input)*	Required only if you want to deploy the solution in an existing VPC that you have created. Ensure that security groups have the permissio ns for an outbound TCP connection.

Parameter	Default	Description
VpcAzs	*(Optional input)*	Required only if you want to deploy the solution in an existing VPC that you have created.
CognitoDomainPrefix	*(Optional input)*	Required only if you want to deploy the solution in an existing Amazon Cognito user pool that you created. If you don't provide a value, the solution generates it.
ExistingCognitoUserPoolId	*(Optional input)*	Required only if you want to deploy the solution in an existing Amazon Cognito user pool that you created.
ExistingCognitoUserPoolClient	*(Optional input)*	Required only if you want to deploy the solution in an existing Amazon Cognito user pool that you created. If you don't provide a value, the solution creates a user pool client. This parameter can only be provided if you provide an **ExistingCognitoUserPoolId** value.

6. Choose **Next**.
7. On the **Configure stack options** page, choose **Next.**
8. On the **Review and create** page, review and confirm the settings. Select the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.
9. Choose **Submit** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a CREATE_COMPLETE status in approximately 10 minutes.

Step 2: Deploy a use case

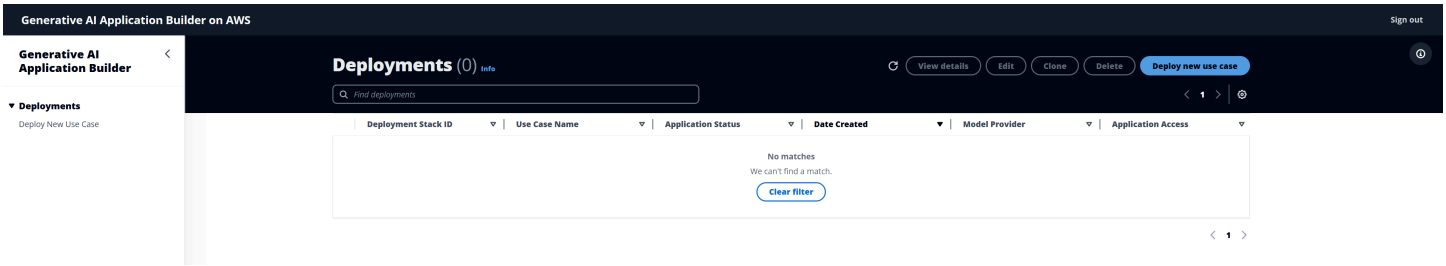
> **Important**
>
> Once the stack has been successfully deployed, a sign-up email is sent to the configured admin user email. Using those credentials, the admin user can sign in to the Deployment dashboard to use the web application.

> **Note**
>
> The DevOps user with access to the AWS Management Console must provide the admin user with the CloudFront URL of the Deployment dashboard UI when the stack completes. The URL can be found in the **Outputs** tab of the CloudFormation stack.

1. Sign in to the Deployment dashboard as an admin user.
2. On the application landing page, choose **Deploy new use case**.

   This launches the deployment wizard, which walks you through building the use case.

Depicts Deployment dashboard landing page - fresh deployment



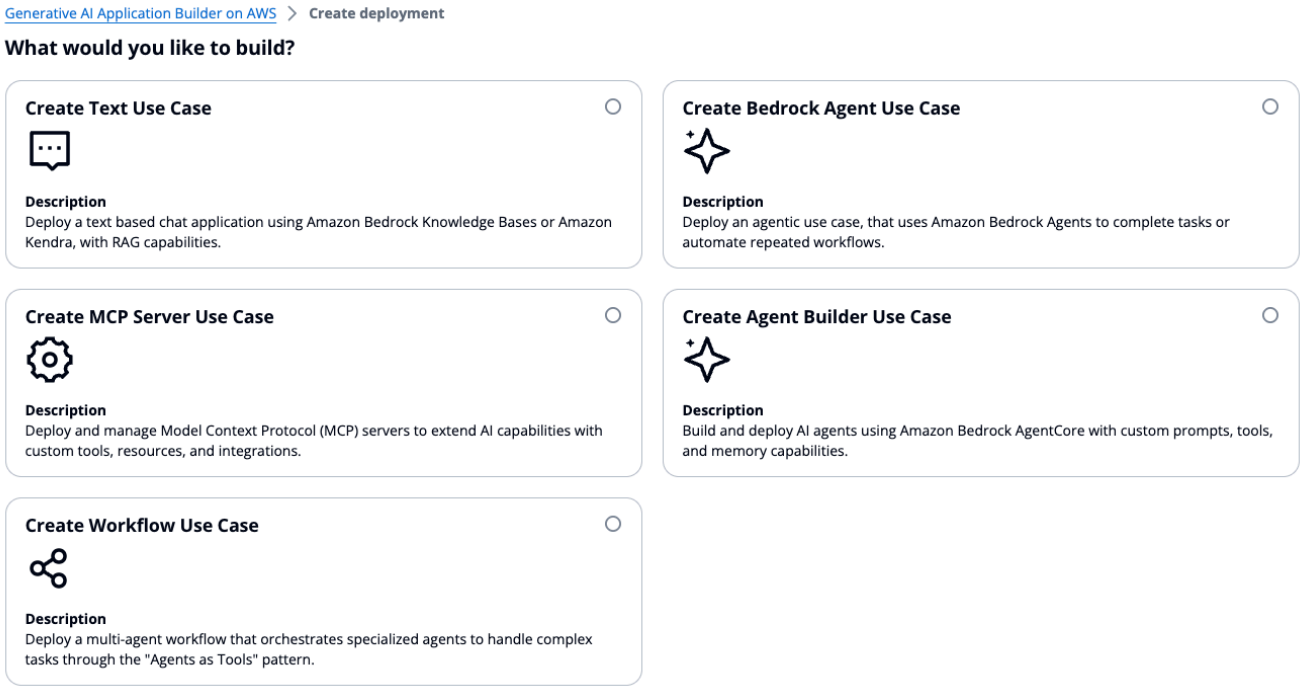
> **Note**
>
> If you need to add additional users to your deployment, refer to the Managing Cognito user pool for more details.

Step 3: Deploy a use case using the Deployment dashboard wizard

In the Deployment dashboard wizard, you must choose between the following:

- Text use case - Deploys a chat application, with optional RAG capabilities
- Bedrock Agent use case - Uses Amazon Bedrock Agents to complete tasks or automate repeated workflows
- MCP Server - Deploy and manage MCP servers with gateway or runtime methods
- Agent Builder - Build and deploy custom agents on AgentCore with MCP integration and memory management
- Workflow Builder - Orchestrate multiple Agent Builder agents using hierarchical delegation

Shows five options: Create Text use case, Create Bedrock Agent use case, Create MCP Server Use Case, Create Agent Builder Use Case, or Create Workflow Use Case.



Step 3a: Deploy a Text use case

This section provides instructions for deploying a Text use case.

Select use case

When you choose **Create Text use case**, the UI opens the **Select use case** screen. Provide the following information:

- Use case name.
- Optional email address for the default user of the use case to be added to the Amazon Cognito user pool for the use case, and to be given permissions to interact with it.
- Whether you want to deploy a UI with this use case. If you don't want to deploy a UI with the use case, you can use the deployed API endpoints for use with your application.

Use case details

The use case details step allows you to configure additional settings for your deployment.

By default, the Text use case creates and configures an Amazon Cognito user pool for you when the solution deploys the Deployment dashboard. The solution authenticates new use cases with a newly created client in the same user pool. However, you can provide an existing user pool ID and client ID in this step if you want to use your own Amazon Cognito user pool and client with the use case.

Important

Admin users have access to all deployed use cases when the Amazon Cognito user pool is created via the deployment wizard. If you provide your own user pool during the deployment, you must ensure that the admin has the permissions to access the deployed use cases.
You will also need to update the Allowed callback URLs and Allowed sign-out URLs in your App clients in Cognito. To do this:

1. Navigate to the Cognito console
2. Choose **User Pools**.
3. Choose your user pool.
4. Choose **App Clients** on the left menu.
5. Choose the app client you want to modify.
6. Choose the **Login pages** tab.
7. Choose **Edit** and add your URLs.

8. Choose **Save changes**.

Additionally, if you need to add more users to a use case, refer to the Managing Cognito user pool section.

Select network configuration

This wizard step allows you to deploy the use case with a pre-existing or new Amazon Virtual Private Cloud (Amazon VPC). If selecting pre-existing VPC, you are required to provide a VPC ID, up to 16 subnet Ids and up to 5 security group IDs to use with this VPC. If you're not using a pre-existing VPC, these settings will be configured for you.

Select model

In the **Select model** step, you can choose your model provider from the dropdown menu. There are two options: **Bedrock** and **SageMaker**.

If you select **SageMaker**, you can create a SageMaker AI model endpoint in the SageMaker AI console and provide the input schema that the model expects and output JSONPath for the LLM response. You can refer to the Using Amazon SageMaker AI as an LLM Provider section and SageMaker AI payload examples provided in the solution's GitHub repository.

If you select **Amazon Bedrock**, you will be presented with four options:

- **Quick Start Models** - Get started quickly with a collection of models with different price/performance characteristics. Recommended for building your first apps. This option allows you to select a model name from the provided list.
- **Other Foundation Models** - Access the full range of foundation models with different capabilities and specializations. This option allows you to enter the model ID for your desired Bedrock on-demand foundation model.
- **Inference Profiles** - Inference profiles leverage Bedrock's cross-region inference to increase throughput and improve resiliency by routing your requests across multiple AWS Regions during peak utilization bursts. This option allows you to enter the ID of the inference profile you want to use.
- **Provisioned Models** - Dedicated throughput capacity for production workloads requiring consistent performance. This option allows you to enter the ARN of the provisioned/custom model to use from Amazon Bedrock.

Model selection step also allows you to choose your advanced model settings. Refer to Advanced LLM settings for details on configuring Amazon Bedrock Guardrails, provisioned throughput for Amazon Bedrock, and additional model parameters.

Cross-region inference

Cross-region inference helps Amazon Bedrock users to seamlessly manage unplanned traffic bursts by using compute across different AWS Regions. To use cross-region inference, you need the *inference profile*. An inference profile is an abstraction over an on-demand pool of resources from a configured set of AWS Regions. It can route your inference request, originating from your source Region, to another Region configured in that pool. This allows traffic distribution across multiple AWS Regions. This helps enable higher throughput and enhanced resilience during periods of peak demands.

Inference profiles are named after the model and Regions that they support. You must call an inference profile from one of the Regions that it includes. For example, as shown in the following table, the inference profile ID `us.anthropic.claude-3-haiku-20240307-v1:0` allows distribution of traffic over `us-east-1` and `us-west-2` Regions of the model you choose. Certain models are only available with an inference profile in a particular Region.

Inference profile	Inference profile ID	Regions included
US Anthropic Claude 3 Haiku	`us.anthropic.claude-3-haiku-20240307-v1:0`	US East (N. Virginia) (`us-east-1`) US West (Oregon) (`us-west-2`)

If you want to use an inference profile ID instead of a model ID, then you must identify the appropriate inference profile ID. See Supported Regions and models for inference profiles in the *Amazon Bedrock User Guide* for more information. In the Amazon Bedrock console, the cross-region inference option in the left navigation menu provides these inference profile IDs.

After you identify the inference profile ID to use, you can use this during the **Select model** stage by performing the following steps:

1. Select **Amazon Bedrock** as the model provider.
2. Select the **Inference Profiles** radio button option.

3. Enter your inference profile ID in the text box that appears.

Refer to Improve resilience with cross-region inference in the *Amazon Bedrock User Guide* for more details on inference profiles.

Select knowledge base

If you're looking to deploy a non-Retrieval Augmented Generation (RAG) use case, you can skip this step.

However, if you wish to enable RAG as a part of your deployment, you can now provide either a pre-configured *Amazon Kendra Index Id* or an *Amazon Bedrock Knowledge Base ID*. You can also create a new Amazon Kendra Index for use with the solution. The solution currently supports Amazon Kendra and Amazon Bedrock Knowledge Bases as knowledge bases for your RAG-based use case deployment.

Refer to the Configuring a Knowledge Base section for guidelines on ingesting data into the knowledge base for use with your RAG-based deployment.

Advanced RAG configurations

The wizard allows you to select advanced options for use with your RAG deployment such as the **number of documents to retrieve** each time a query is sent to your knowledge base, a **static text response** from the LLM when no documents are found in the knowledge base, whether you wish to **display document sources** with your LLM response for sanity checks, etc. You can additionally also configure knowledge base specific configurations for Amazon Kendra such as Role-based Access Control (RBAC), or Override Search Type when using Amazon OpenSearch Serverless with Amazon Bedrock Knowledge Bases. Refer to the Advanced Knowledge Base settings section for more details on these advanced settings.

> **Note**
>
> Your knowledge base must be in the same account and Region as the deployed Deployment dashboard and use case stacks.

Select prompts and token limits

In this step, you can configure your prompt for use with the LLM. Prompts may require placeholders such as `{input}`, `{history}` and `{context}`. These placeholders instruct the LLM

on where to draw user input, conversation history, and information retrieved from the knowledge base from.

- For Bedrock model provider, the system prompt must be provided which has no restrictions for a non-RAG use-case. The disambiguation prompt for Bedrock model provider however, requires a minimum of two placeholders - `{input}` and `{history}`
- For SageMaker model provider, system and disambiguation prompts, both require a minimum of two placeholders - `{input}` and `{history}`.
- For RAG use cases, for each model provider, the `{context}` placeholder is additionally required.

For more information, see Configuring your prompts. You can also refer to the Tips for managing model token limits section while selecting token limit sizes for your prompts.

Enable multimodal input

This step allows you to enable multimodal input capabilities for your use case. When enabled, users can upload and send images and documents along with their text queries.

Supported file types and constraints:

- **Images:** Up to 20 images per message. Each image must be no more than 3.75 MB in size and 8,000 px in height and width. Supported formats: png, jpeg, gif, webp
- **Documents:** Up to 5 documents per message. Each document must be no more than 4.5 MB in size. Supported formats: pdf, csv, doc, docx, xls, xlsx, html, txt, md

How to use multimodal input:

1. Enable the **MultimodalEnabled** parameter during use case deployment
2. In the chat interface, users can upload files in two ways:
   - Clicking the upload button in the chat input box, or
   - Dragging and dropping files directly into the chat interface
3. Files are uploaded to Amazon S3 and processed by the selected model
4. Uploaded files are automatically deleted after 48 hours

File status tracking:

DevOps users can monitor file metadata in DynamoDB, which includes upload time and processing status. Files can have the following statuses:

- **pending** - File upload has been initiated but not yet completed. This is the initial status when a presigned URL is generated.
- **uploaded** - File has been successfully uploaded to S3 and is ready for processing by the model.
- **deleted** - File has been deleted by the user and should no longer be accessible for processing.
- **invalid** - File failed validation checks (for example, file type mismatch or security validation failure).

Files in **pending** status that are never uploaded will be automatically cleaned up when their TTL expires. Only files with **uploaded** status can be processed by the model.

The S3 multimodal bucket and DynamoDB metadata table is available in the Deployment Dashboard outputs with the keys `MultimodalDataBucketName` and `MultimodalDataMetadataTable`, respectively.

> **Note**
>
> Not all models support multimodal input. Ensure your selected model supports image and document processing before enabling this feature. Refer to the Supported foundation models in Amazon Bedrock documentation to check which model support Image as an input modality.

> **Important**
>
> Files uploaded by users are stored in Amazon S3 with a 48-hour lifecycle policy. Metadata about uploaded files is stored in Amazon DynamoDB with a 24-hour TTL for conversation history.

Review and deploy

After this step, review the settings you selected and choose **Deploy Use Case**. The new use case then deploys and becomes visible in your Deployment dashboard view to manage further.

Step 3b: Deploy a Bedrock Agent use case

The Bedrock Agent use case provides a powerful and secure mechanism for invoking Amazon Bedrock Agents within your use cases. This feature allows developers to seamlessly integrate the capabilities of AI-powered autonomous agents that can orchestrate and execute multi-step tasks across various foundation models, data sources, software applications, and user conversations while maintaining robust security measures.

Prerequisites

Before creating an Amazon Bedrock agent, ensure that you have the following:

1. The AWS account where Generative AI Application Builder on AWS is deployed, with an access to the Amazon Bedrock console.
2. Appropriate IAM permissions to create and manage Amazon Bedrock Agents.

Creating an Amazon Bedrock Agent

Refer to the Create and configure agent manually in the *Amazon Bedrock User Guide* for detailed instructions on creating an agent. You can configure options such as:

- Instructions (prompts) for your agent
- Knowledge base, which is used to look up additional information based on user's input
- Agent's memory to allow agents to remember information across multiple sessions (for a maximum of 30 days)

After you successfully create an Amazon Bedrock agent, you can proceed to the Generative AI Application Builder on AWS Bedrock Agent use case wizard flow. To do so, choose **Deploy a new use case** on the Deployment dashboard and select **Create Bedrock Agent Use Case**. Follow the wizard and use the following steps to configure the use case.

Select use case

This step is the same as the Text use case described previously.

Select network configuration

This step is the same as the Text use case described previously

Select agent

In this step, you must provide the **Agent ID** and **Alias ID** of the Amazon Bedrock agent that you created.

Step 3c: Deploy an MCP Server use case

The MCP (Model Context Protocol) Server use case enables you to deploy and manage MCP servers that can be integrated with AI models and agents. MCP servers provide a standardized way to expose tools, resources, and capabilities to AI applications. You can either create MCP servers from existing Lambda functions and APIs, or host custom MCP servers using container images.

Prerequisites

Before deploying an MCP Server use case, ensure that you have the following:

1. The AWS account where Generative AI Application Builder on AWS is deployed.
2. Appropriate IAM permissions to create and manage Amazon Bedrock AgentCore resources.
3. Depending on your chosen creation method:
   - **For Gateway method (Lambda/API/MCP Server)**: Lambda functions, API endpoints with their corresponding schema files (JSON format for Lambda, OpenAPI/Smithy for APIs), or MCP Server URL endpoints
   - **For Runtime method (ECR)**: A Docker container image pushed to Amazon ECR containing your MCP server implementation

MCP Server creation methods

The solution supports two methods for creating MCP servers:

Create from Lambda, API, or MCP Server (Gateway method)

This method creates an MCP gateway that wraps existing Lambda functions, REST APIs, or external MCP servers, making them accessible as MCP tools. The gateway handles protocol translation between MCP and your existing services.

- **Lambda targets**: Integrate existing Lambda functions by providing the function ARN and a JSON schema file describing the function's input/output format
- **OpenAPI targets**: Integrate REST APIs using OpenAPI specifications (JSON or YAML format) with support for OAuth 2.0 or API Key authentication

- **Smithy targets**: Integrate APIs defined using Smithy model files (.smithy or .json format)
- **MCP Server targets**: Connect directly to external MCP servers via URL endpoints, allowing integration of existing MCP servers without deploying new infrastructure

You can configure multiple targets (up to 10) within a single MCP gateway, each representing a different tool or capability.

Hosting from ECR Image (Runtime method)

This method deploys a containerized MCP server from an Amazon ECR image. Use this approach when you have a custom MCP server implementation that needs to run as a standalone service.

- Provide the ECR image URI (must include a tag, e.g., `:latest` or `:v1.0.0`)
- Optionally configure environment variables to pass configuration to your container
- The container must implement the MCP protocol and expose the required endpoints

Deploying an MCP Server

To deploy an MCP Server use case, choose **Deploy a new use case** on the Deployment dashboard and select **Create MCP Server Use Case**. Follow the wizard and use the following steps to configure the use case.

Select use case

This step is the same as the Text use case described previously.

Select network configuration

Currently only public access is enabled and VPC is not supported for netwrok configuation.

Create MCP Server

In this step, you configure your MCP server deployment:

MCP server creation method

Choose between the two creation methods:

- **Create from Lambda, API, or MCP Server**: Create an MCP gateway from existing Lambda functions, API specifications, or external MCP server endpoints
- **Hosting from ECR Image**: Deploy a custom MCP server from a container image

> **Note**
>
> The creation method cannot be changed after deployment. If you need to switch methods, you must deploy a new MCP Server use case.

Gateway Configuration (for Lambda/API/MCP Server method)

If you selected the Gateway method, configure one or more targets:

1. **Target name** (required): A friendly name to identify this target configuration
2. **Target description** (optional): A brief description of what this target does
3. **Target Type**: Select the type of target to configure:
   - **Lambda**: For AWS Lambda functions
   - **OpenAPI**: For REST APIs with OpenAPI specifications
   - **Smithy**: For APIs with Smithy model definitions
   - **MCP Server**: For direct connection to external MCP servers via URL endpoints
4. **Schema File** (required): Upload the schema file that describes your target:
   - For Lambda: JSON schema file describing input/output format. For details on creating Lambda tool schemas, see Lambda tool schema in the *Amazon Bedrock AgentCore Developer Guide*.
   - For OpenAPI: OpenAPI specification file (JSON or YAML). For details on OpenAPI schema requirements, see OpenAPI schema in the *Amazon Bedrock AgentCore Developer Guide*.
   - For Smithy: Smithy model file (.smithy or .json). For details on building Smithy targets, see Building Smithy targets in the *Amazon Bedrock AgentCore Developer Guide*.
5. **Lambda Function ARN** (required for Lambda targets): The ARN of the Lambda function to integrate
6. **MCP Server URL** (required for MCP Server targets): The URL endpoint of the external MCP server to connect to. The URL must be properly encoded and the MCP server must support tool capabilities with MCP protocol versions 2025-06-18. For more information, see MCP servers targets in the *Amazon Bedrock AgentCore Developer Guide*.
7. **Outbound Authentication** (required for OpenAPI targets): Configure authentication for REST API calls:
   - **Authentication Type**: Choose OAuth 2.0 or API Key

- **Outbound Auth Provider ARN**: The ARN of the credential provider in Amazon Bedrock AgentCore token vault
- **Additional configurations**: Depending on authentication type:
  - For OAuth 2.0: Configure scopes and custom parameters
  - For API Key: Specify location (header or query parameter), parameter name, and optional prefix

You can add multiple targets (up to 10) by choosing **Add another target**. Each target represents a separate tool or capability exposed by your MCP server.

ECR Configuration (for ECR Image method)

If you selected the Runtime method, provide:

1. **ECR Image URI** (required): The full URI of your Docker image in Amazon ECR
   - Format: `account-id.dkr.ecr.region.amazonaws.com/repository-name:tag`
   - The image must be in the same AWS Region as your deployment
   - A tag is required (e.g., `:latest`, `:v1.0.0`)
2. **Environment variables** (optional): Configure key-value pairs to pass to your container at runtime
   - Use these to provide configuration, credentials, or custom flags
   - You can add up to 10 environment variables

Review and deploy

After configuring your MCP server, review the settings you selected and choose **Deploy Use Case**. The new MCP Server use case then deploys and becomes visible in your Deployment dashboard view for further management.

> **Note**
>
> MCP Server deployments create resources in Amazon Bedrock AgentCore, including gateways, runtimes, and workload identities. These resources are automatically managed by the solution and will be cleaned up when you delete the use case.

Step 3d: Deploy an Agent Builder use case

The Agent Builder enables you to create, configure, and deploy production-ready AI agents on Amazon Bedrock AgentCore. This feature provides full control over agent behavior through system prompts, model selection, MCP server integration, and memory management.

The deployment process is primarily the same as for a Text use case, with some notable differences.

Select use case

This step is the same as the Text use case described previously.

Use case details

This step is the same as the Text use case described previously.

Configure agent

In this step, you configure the core agent settings including system prompt, available MCP servers/ Strands tools, and memory.

System Prompt

The system prompt defines the agent's behavior, personality, and capabilities. You can:

- Edit the default system prompt template
- Use the **Reset to default** button to restore the original template
- Include instructions for tool usage and response formatting

MCP Server Integration (Optional)

Configure Model Context Protocol servers to provide your agent with access to enterprise tools and data:

1. Select from available MCP servers in the dropdown
2. Review available out of the box tools that will be accessible to the agent

> **Note**
>
> MCP servers must be configured and accessible before deployment. Refer to the MCP documentation for server setup instructions.

Memory Configuration

Configure how the agent maintains context and knowledge:

- **Short-term Memory**: Enabled by default for all agents. Maintains conversation context within sessions.
- **Long-term Memory**: Toggle to enable extraction and storage of insights across sessions. Uses AgentCore Memory with semantic memory strategy.

Review and deploy

After this step, review the settings you selected and choose **Deploy Use Case**. The Agent Builder deployment typically completes in 10-15 minutes. The new use case then becomes visible in your Deployment dashboard view to manage further.

Step 3e: Deploy a Workflow use case

The Workflow Builder enables you to create supervisor agents that orchestrate multiple Agent Builder agents using the Agents as Tools delegation pattern. This feature allows you to build complex multi-agent workflows by reusing existing Agent Builder deployments.

The deployment process follows a similar pattern to Agent Builder, with additional steps for agent discovery and selection.

Select use case

This step is the same as the Text use case described previously.

Use case details

This step is the same as the Text use case described previously.

Configure supervisor agent

In this step, you configure the supervisor agent that will coordinate the specialized Agent Builder agents.

System Prompt

The system prompt defines how the supervisor agent delegates work to specialized agents. You can:

- Edit the default system prompt template
- Include instructions for agent selection and delegation
- Define how to aggregate results from multiple agents
- Use the **Reset to default** button to restore the original template

> **Note**
>
> The system prompt should clearly describe when and how to use each specialized agent. Agent descriptions are critical for proper delegation.

Model Selection

Select the foundation model for the supervisor agent. The supervisor agent uses this model to:

- Understand user requests
- Select appropriate specialized agents
- Coordinate agent execution
- Aggregate and format responses

Select specialized agents

In this step, you select which Agent Builder agents the supervisor can delegate work to.

Adding Agents

1. Click **Add Agent** to open the agent selection dialog

2. Select one or more Agent Builder agents from the list
3. Review the agent descriptions that will be provided to the supervisor
4. Confirm the selection

> **Note**
>
> - Workflows require at least 1 Agent Builder use case as a specialized agent
> - All specialized agents must be successfully deployed before creating the workflow

Review and deploy

Review the workflow configuration including:

- Supervisor agent system prompt and model
- List of specialized agents
- Memory settings

Choose **Deploy Use Case**. The Workflow deployment typically completes in 15-20 minutes. The new workflow becomes visible in your Deployment dashboard view to manage further.

Step 4: Post-deployment configuration

This section provides recommendations for configuring the solution after deployment.

Amazon S3 bucket versioning, lifecycle policies, and cross-Region replication

This solution doesn't enforce lifecycle configurations on the buckets it creates. We recommend the following:

- Setting lifecycle configurations for production deployments. For details, see Setting lifecycle configuration on a bucket in the *Amazon Simple Storage Service User Guide*.
- Enabling versioning and cross-Region replication for Amazon S3 buckets based on the use case for which the solution is deployed.

Amazon DynamoDB backups

This solution uses DynamoDB for several purposes (see AWS services in this solution). The solution doesn't enable backups for the tables it creates. We recommend creating a backup of this feature for production deployments. See Backing up a DynamoDB table and Using AWS Backup for DynamoDB for details.

Amazon CloudWatch dashboard and alarms

The solution deploys a custom dashboard in CloudWatch to render charts from custom published metrics and AWS service metrics. We recommend creating CloudWatch alarms and adding notifications based on the use case for which the solution is deployed.

Amazon CloudWatch Logs

Lambda logs are configured to never expire and API Gateway logs are configured with a 10-year expiry. You can update the expiry of the respective log groups to align with your enterprise's record retention policy.

Custom web domains with TLS v1.2 or higher certificates

The solution deploys a web UI and Edge Optimized API Gateway using CloudFront. CloudFront's domain doesn't enforce TLS v1.2 or higher certificates. We recommend creating a custom domain using Amazon Route 53, creating a certificate using AWS Certificate Manager, or using an existing certificate if your organization has one.

For additional details, refer to the Amazon Route 53 Developer Guide and Choosing a minimum TLS version for a custom domain in API Gateway.

Scaling with Amazon Kendra

This solution provides the ability to use Amazon Kendra to perform NLP-powered intelligent search across the ingested documents. You can increase the capacity of Amazon Kendra using the following CloudFormation parameters for larger workloads:

Parameter	Default	Description
Amazon Kendra additional query capacity	`0`	The amount of extra query capacity for an index and

Parameter	Default	Description
		GetQuerySuggestions capacity. An additional capacity unit for an index provides approximately 8,000 queries per day.
Amazon Kendra additional storage capacity	`0`	The amount of extra storage capacity for an index. A single capacity unit provides 30 GB of storage space or 100,000 documents, whichever reaches first.
Amazon Kendra edition	`Developer`	Amazon Kendra provides Developer and Enterprise Editions to create indexes. For more information about the differences between Amazon Kendra Editions, see Amazon Kendra pricing.

To modify the values of these CloudFormation parameters, select the appropriate values at the time of stack deployment. For more information on query and storage capacity units, see Adjusting capacity.

> **Note**
>
> If the Text use case is not deployed with RAG enabled, then an Amazon Kendra index is not used or created.

Setting up SSO using Idp federation

This solution allows integration with external identity providers that support SAML or OIDC based identity federation. When the solution deploys, it creates an Amazon Cognito user pool and

individual app client integration for the Deployment dashboard and individual use cases. Based on the external Idp, follow the steps provided in the Configuring identity providers for your user pool section of the *Amazon Cognito Developer Guide* and choose the app client integration for the Deployment dashboard or use case you would like to setup SSO with.

To pass the user group information to knowledge base or vector stores in a RAG based architecture, you will need to map user groups from the external Idp to Amazon Cognito user groups. The solution provides an initial scaffolding Lambda function trigger to be mapped with the pre token generation phase. The Lambda function has the group_mapping.json file which must be updated to provide the group mappings. Refer to Customizing user pool workflows with Lambda triggers for Lambda triggers supported by Amazon Cognito.

Manual User Pool configuration

If you choose not to pass an Admin or default user email during deployment, you must manually create the appropriate user groups in Amazon Cognito to ensure correct permissions:

1. For the Deployment dashboard, create a group named `Admin` in your Cognito user pool.
2. For each use case, create a group named `${UseCaseName}-Users` in your Cognito user pool, where `${UseCaseName}` is the name of your deployed use case.

These groups are required for the authorization mechanism to work correctly. Any users you want to grant access to must be added to the appropriate groups.

If `placeholder@example.com` is passed, the Cognito group will be created, but you must still create the associated users and assign them to the group.

Customizing login screen

This solution uses Amazon Cognito hosted UI to render the login page. To customize the built-in sign-in page, refer to Customizing the built-in sign-in and sign-up webpages in the *Amazon Cognito Developer Guide*.

Additional security considerations

Based on the use case for which you deploy the solution, review the following security recommendations:

- **Customer managed AWS KMS encryption keys** - The solution uses AWS managed AWS KMS keys by default, since these are available at no additional cost. Review your use case to determine if you should update the solution to use customer managed AWS KMS keys.
- **API Gateway throttling rules** - The solution deploys with default throttling rules on API Gateway. Based on your use case and expected transaction volumes, we recommend that you configure throttling for the APIs. For details, see Throttle API requests for better throughput in the *Amazon API Gateway Developer Guide*.
- **Enabling AWS CloudTrail** - As a recommended security practice, consider enabling AWS CloudTrail in the AWS account where the solution is deployed to log API calls in the AWS account. For details, see the AWS CloudTrail User Guide.
- **Drift detection** - We recommend configuring drift detection on CloudFormation stacks to identify and be notified of unintentional or malicious changes to the deployed solution stack. For details, see Implementing an alarm to automatically detect drift in AWS CloudFormation stacks.
- **Cognito JSON Web Tokens (JWTs)** - The solution uses Amazon Cognito-issued JWTs to authenticate with the REST API endpoints. We configured the solution with a five-minute expiry for ID tokens and access tokens. When a user logs out, their ability to generate new tokens is revoked (refresh token is revoked). However, until the expiry of the current token, any requests to the API endpoint will be successfully authenticated, since they have a valid token. Review the security considerations for your use case and adjust the token validity period.

Customizing lifecycle policies:

For production deployments, review and adjust the lifecycle policies based on your retention requirements. See Setting lifecycle configuration on a bucket in the *Amazon Simple Storage Service User Guide*.

Multimodal file storage and lifecycle

If you enabled multimodal input capabilities (**MultimodalEnabled** set to `Yes`) for your use case, the solution creates an Amazon S3 bucket to store uploaded files and a DynamoDB table to track file metadata.

Default lifecycle policies:

- **S3 files:** Automatically deleted after 48 hours
- **DynamoDB metadata:** Records expire after 24 hours (conversation history TTL)

Security considerations:

- Files are partitioned by use case ID, user ID, conversation ID and message ID and a file is stored with a UUID name instead. The mapping for the UUID to file names is available in the DynamoDB metadata table
- Users can only access files they uploaded within their own conversations
- File type validation is performed using magic number detection
- We recommend enabling Amazon GuardDuty Malware Protection for S3 to scan uploaded files for malicious content

Deploying a standalone Text use case

Follow the step-by-step instructions in this section to configure and deploy the solution into your account.

Time to deploy: Approximately 10-30 minutes

1. Sign in to the AWS Management Console and select the button to launch the CloudFront template that you want to deploy.

BedrockChat.template	Launch solution
SageMakerChat.template	Launch solution

2. The template launches in the US East (N. Virginia) Region by default. To launch the solution in a different AWS Region, use the Region selector in the console navigation bar.

   Note: This solution uses Amazon Kendra and Amazon Bedrock, which are not currently available in all AWS Regions. If using these features, you must launch this solution in an AWS Region where these services are available. For the most current availability by Region, see the AWS Regional Services List.

3. On the **Create stack *page, verify that the correct template URL is in the *Amazon S3 URL *text box and choose *Next**.

4. On the *Specify stack details *page, assign a name to your solution stack. For information about naming character limitations, see IAM and STS Limits in the *AWS Identity and Access Management User Guide*.
5. Under **Parameters**, review the parameters for this solution template and modify them as necessary. This solution uses the following default values.

UseCaseUUID	*<_Requires input_>*	36 character long UUIDv4 to identify this deployed use case within an application.
UseCaseConfigRecordKey	*<_Requires input_>*	Key corresponding of the record containing configura tions required by the chat provider Lambda at runtime. The record in the table must have a **key** attribute matching this value, and a **config** attribute containin g the desired configuration. This record will be populated by the deployment platform if in use. For standalone deployments of this use case, a manually created entry in the table defined in **UseCaseConfigTableName** is required.
UseCaseConfigTableName	*<_Requires input_>*	The stack will read the configuration from the table with this name at the key **UseCaseConfigRecordKey**
ExistingRestApiId	*(Optional input)*	Existing API Gateway REST API ID to use. If not provided, a new API Gateway REST API will be created. Typically

		provided when deploying from the Deployment dashboard. Note: Using Existing APIs can help reduce resource duplication and simplify management of APIs when you need to deploy multiple standalone use cases. When supplying existing APIs for a standalone use case, you are responsible for ensuring that the API is configured with the required route(s) with expected models. A required pre-configured /details route (fetches use case details during chat) and optionally, a /feedback route (if **FeedbackEnabled** is set to `Yes` to enable collection of feedback for LLM chat responses) must be configured. Additionally, **ExistingApiRootResourceId**, **ExistingCognitoUserPoolId** and **ExistingCognitoGroupPolicyTableName** must also be provided.

ExistingApiRootResourceId	*(Optional input)*	Existing API Gateway REST API Root Resource ID to use. REST API Root Resource ID can be obtained from the AWS console by selecting the root resource (/) in the "Resources" section of the API. The Resource ID will then be displayed in the Resource details panel. You can alternatively run a describe API call on your REST API to find the Root Resource ID.
FeedbackEnabled	No	If set to No, the deployed use case stack will not have access to the feedback feature.
ExistingModelInfoTableName	*(Optional input)*	DynamoDB table name for the table which contains model info and defaults. Used by the deployment platform. If omitted, a new table will be created to house model defaults.

DefaultUserEmail	`placeholder@example.com`	Email of the default user for this use case. An Amazon Cognito user for this email is created to access the use case. If not provided, the Cognito Group and User will not be created. You may also use `placeholder@example.com` to create the Group but not the User. Refer to Manual User Pool Configuration for information on setting up your user pool.
ExistingCognitoUserPoolId	*(Optional input)*	**UserPoolId** of an existing Amazon Cognito user pool which this use case will be authenticated with. Typically provided when deploying from the Deployment dashboard, but can be omitted when deploying this use case stack standalone.
CognitoDomainPrefix	*(Optional input)*	Enter a value if you want to provide a domain for the Cognito User Pool Client. If you don't provide a value, the deployment will generate one.

ExistingCognitoUserPoolClient	*(Optional input)*	Provide a User Pool Client (App Client) to use an existing one. If you don't provide a User Pool Client, a new one will be created. This parameter can only be provided if an existing User Pool Id is provided.
ExistingCognitoGroupPolicyTableName	*(Optional input)*	Name of the DynamoDB table containing user group policies. This is used by the custom authorizer on the use case's API. Typically, you can provide an input when deploying from the deployment platform, but can be omitted when deploying this use case stack standalone.
RAGEnabled	`true`	If set to true, the deployed use case stack uses the provided Amazon Kendra index created to provide RAG functionality. If set to `false`, the user interacts directly with the LLM.
KnowledgeBaseType	Bedrock	Knowledge base type to be used for RAG. Only set if RAGEnabled is `true`. Can be Bedrock or Kendra. Note: Only relevant if **RAGEnabled** is true.

ExistingKendraIndexId	*(Optional input)*	Index ID of an existing Kendra index to be used for the use case. If none is provided and **Knowledge BaseType** is Kendra, a new index will be created for you. Note: Only relevant if **RAGEnabled** is `true` and **KnowledgeBaseType** is `Kendra`.
NewKendraIndexName	*(Optional input)*	Name for the new Kendra index to be created for this use case. Only applies if **ExistingKendraIndexId** is not supplied. Note: Only relevant if **RAGEnabled** is true and **KnowledgeBaseType** is Kendra.
NewKendraQueryCapacityUnits	0	Additional query capacity units for the new Amazon Kendra index to be created for this use case. Only applies if **ExistingKendraIndexId** is not supplied, see CapacityUnitsConfiguration. Note: Only relevant if **RAGEnabled** is `true` and **KnowledgeBaseType** is `Kendra`.

NewKendraStorageCapacityUnits	`0`	Additional storage capacity units for the new Amazon Kendra index to be created for this use case. Only applies if **ExistingKendraIndexId** is not supplied, see CapacityUnitsConfiguration. Note: Only relevant if **RAGEnabled** is `true` and **KnowledgeBaseType** is `Kendra`.
NewKendraIndexEdition	*(Optional input)*	The edition of Amazon Kendra to use for the new Amazon Kendra index to be created for this use case. Only applies if **ExistingKendraIndexId** is not supplied, see Amazon Kendra Editions. Note: Only relevant if **RAGEnabled** is `true` and **KnowledgeBaseType** is `Kendra`.

BedrockKnowledgeBaseId	*(Optional input)*	ID of the bedrock knowledge base to use in a RAG use case. Cannot be provided if **ExistingKendraIndexId** or **NewKendraIndexName** are provided. Note: Only relevant if **RAGEnabled** is `true` and **KnowledgeBaseType** is `Bedrock`.
VpcEnabled	No	Should the stacks resources be deployed within a VPC.
CreateNewVpc	No	Select `Yes`, if you want the solution to create a new VPC for you and be used for this use case. Note: Only relevant if **VpcEnabled** is `Yes`.
IPAMPoolId	*(Optional input)*	If you want to assign the CIDR range using Amazon VPC IP Address Manager, provide the IPAM pool Id to use. Note: Only relevant if **VpcEnabled** is `Yes` and **CreateNewVpc** is No.

ExistingVpcId	*(Optional input)*	VPC ID of an existing VPC to be used for the use case. Note: Only relevant if **VpcEnabled** is Yes and **CreateNewVpc** is No.
ExistingPrivateSubnetIds	*(Optional input)*	Comma separated list of subnet IDs of existing private subnets to be used to deploy the Lambda function. Note: Only relevant if **VpcEnabled** is Yes and **CreateNewVpc** is No.
ExistingSecurityGroupIds	*(Optional input)*	Comma separated list of security groups of the existing VPC to be used for configuring Lambda functions. Note: Only relevant if **VpcEnabled** is Yes and **CreateNewVpc** is No.
VpcAzs	*(Optional input)*	Comma separated list of AZs in which subnets of the VPCs are created Note: Only relevant if **VpcEnabled** is Yes and **CreateNewVpc** is No.

UseInferenceProfile	No	If the model configured is Bedrock, you can indicate if you are using Bedrock Inference Profile. This will ensure that the required IAM policies will be configured during stack deployment. For more details, refer to the following https://docs.aws.amazon.com/bedrock/latest/userguide/cross-region-inference.html
DeployUI	Yes	Select the option to deploy the frontend UI for this deployment. Selecting No, will only create the infrastructure to host the APIs, the authentication for the APIs, and backend processing.

6. Choose **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review and confirm the settings. Select the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.
9. Choose **Create stack** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a CREATE_COMPLETE status in approximately 10-30 minutes.

Deploying a standalone Bedrock Agent use case

Follow the step-by-step instructions in this section to configure and deploy the solution into your account.

Time to deploy: Approximately 10-30 minutes

1. Sign in to the AWS Management Console and select the button to launch the CloudFront template.

BedrockAgent.template	

2. The template launches in the US East (N. Virginia) Region by default. To launch the solution in a different AWS Region, use the Region selector in the console navigation bar.

> **Note**
>
> This solution uses Amazon Bedrock, which is not currently available in all AWS Regions. If you're using these features, you must launch this solution in an AWS Region where these services are available. For the most current availability by Region, see the AWS Regional Services List.

3. On the **Create stack** page, verify that the correct template URL is in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, assign a name to your solution stack. For information about naming character limitations, see {https---docs-aws-amazon-com-https---docs-aws-amazon-com-IAM-latest-UserGuide-reference-iam-limits-html}[IAM and AWS STS quotas] in the *AWS Identity and Access Management User Guide*.
5. Under **Parameters**, review the parameters for this solution template and modify them as necessary. This solution uses the following default values.

Parameter	Default entry	Description
UseCaseUUID	*<_Requires input_>*	36 character long UUIDv4 to identify this deployed use case within an application.
UseCaseConfigRecordKey	*<Requires input>*	Key corresponding to the record that contains configurations required by the chat provider Lambda function at runtime.

Parameter	Default entry	Description
		The record in the table must have a **key** attribute matching this value, and a **config** attribute containing the desired configuration. This record will be populated by the deployment platform if it's in use. For standalone deployments of this use case, a manually created entry in the table defined in **UseCaseConfigTableName** is required.
UseCaseConfigTableName	*<Requires input>*`	The stack will read the use case configuration from the table provided here and using the record key defined in **UseCaseConfigRecordKey**.
DefaultUserEmail	`placeholder@example.com`	Email of the default user for this use case. The solution creates an Amazon Cognito user for this email to access the use case.

Parameter	Default entry	Description
ExistingRestApiId	*(Optional input)*	Existing API Gateway REST API ID to use. If not provided, a new API Gateway REST API will be created. Typically provided when deploying from the Deployment dashboard. Note: Using Existing APIs can help reduce resource duplication and simplify management of APIs when you need to deploy multiple standalone use cases. When supplying existing APIs for a standalone use case, you are responsible for ensuring that the API is configured with the required route(s) with expected models. A required pre-configured / details route (fetches use case details during chat) and optionally, a /feedback route (if **FeedbackEnabled** is set to `Yes` to enable collection of feedback for LLM chat responses) must be configured. Additionally, **ExistingApiRootResourceId**, **ExistingCognitoUserPoolId** and **ExistingCognitoGroupPolicyTableName** must also be provided.

Parameter	Default entry	Description
ExistingApiRootResourceId	*(Optional input)*	Existing API Gateway REST API Root Resource ID to use. REST API Root Resource ID can be obtained from the AWS console by selecting the root resource (/) in the "Resources" section of the API.The Resource ID will then be displayed in the Resource details panel. You can alternatively run a describe API call on your REST API to find the Root Resource ID.
FeedbackEnabled	No	If set to No, the deployed use case stack will not have access to the feedback feature.
CognitoDomainPrefix	*(Optional input)*	Enter a value if you want to provide a domain for the Amazon Cognito user pool client. If you don’t provide a value, the solution generates one.

Parameter	Default entry	Description
ExistingCognitoUserPoolId	*(Optional input)*	**UserPoolId** of an existing Amazon Cognito user pool that you want to authentic ate this use case with. NOTE: You typically provide this ID when deploying from the Deployment dashboard , but you can omit it when deploying this use case stack standalone.
ExistingCognitoUse rPoolClient	*(Optional input)*	Provide a user pool client (app client) to use an existing one. If you don't provide a user pool client, the solution creates one. You can only provide this parameter if you provided an **ExistingC ognitoUserPoolId**.
ExistingCognitoGro upPolicyTableName	*(Optional input)*	Name of the DynamoDB table containing user group policies. This is used by the custom authorizer on the use case's API. NOTE: You typically provide this name when deploying from the Deployment dashboard, but you can omit it when deploying this use case stack standalone.
VpcEnabled	No	Whether the stacks resources be deployed within a VPC.

Parameter	Default entry	Description
CreateNewVpc	No	Select Yes if you want the solution to create a new VPC for you and to use it for this use case. NOTE: This parameter is only relevant if **VpcEnabled** is Yes.
IPAMPoolId	*(Optional input)*	If you want to assign the CIDR range using IPAM, provide the IPAM pool ID to use. NOTE: This parameter is only relevant if **VpcEnabled** is Yes and **CreateNewVpc** is No.
ExistingVpcId	*(Optional input)*	VPC ID of an existing VPC to be used for the use case. NOTE: This parameter is only relevant if **VpcEnabled** is Yes and **CreateNewVpc** is No.
ExistingPrivateSubnetIds	*(Optional input)*	Comma separated list of subnet IDs of existing private subnets to be used to deploy the Lambda function. NOTE: This parameter is only relevant if **VpcEnabled** is Yes and **CreateNewVpc** is No.

Parameter	Default entry	Description
ExistingSecurityGroupIds	*(Optional input)*	Comma-separated list of security groups of the existing VPC to be used for configuring Lambda functions. NOTE: This parameter is only relevant if **VpcEnabled** is `Yes` and **CreateNewVpc** is No.
VpcAzs	*(Optional input)*	Comma separated list of AZs in which subnets of the VPCs are created Note: Only relevant if **VpcEnabled** is `Yes` and **CreateNewVpc** is No.
BedrockAgentId	*<Requires input>*	The ID of the Amazon Bedrock Agent to be used.
BedrockAgentAliasId	*<Requires input>*	The alias ID of the Amazon Bedrock Agent to be used.
DeployUI	`Yes`	Select the option to deploy the frontend chat UI for this deployment. Selecting No results in creating the infrastructure to host the APIs, the authentication for the APIs, and backend processing without the chat UI.

6. Choose **Next**.
7. On the **Configure stack options** page, choose **Next**.

8. On the **Review** page, review and confirm the settings. Select the box acknowledging that the template will create IAM resources.
9. Choose **Create stack** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a CREATE_COMPLETE status in approximately 10-30 minutes.

Supplying a DynamoDB chat configuration

When deploying a use case, **UseCaseConfigRecordKey** and **UseCaseConfigTableName** are required CloudFormation parameters which are normally populated by the Deployment dashboard. The deployment dashboards stack handles the creation and configuration of this table, while calls to the deployment API trigger population of the parameters.

When performing a standalone deployment, you must do the following:

1. Create a DynamoDB table with a hash key of **key**.
2. Create a record in the table containing the configuration for the use case as a record of the format: `{key: some_use_case_key, config: {your_configuration}.`
3. Pass the chosen **UseCaseConfigTableName** and **UseCaseConfigRecordKey** (`some_use_case_key` in this example) parameters to the use case stack when deploying.

To create a suitable configuration for a standalone deployment, you can create a required use case from the Deployment dashboard, and copy record from the configuration table. Otherwise, you can craft your own configuration based on the following example for a Bedrock deployment:

```
{
 "UseCaseName": "SampleUseCase",
 "ConversationMemoryParams": {
 "ConversationMemoryType": "DynamoDB",
 "HumanPrefix": "H",
 "AiPrefix": "A",
 "ChatHistoryLength": 20
 },
 "KnowledgeBaseParams": {
 "KnowledgeBaseType": "Bedrock",
 "NumberOfDocs": 2,
 "ScoreThreshold": 0,
```

```
 "ReturnSourceDocs": false,
 "BedrockKnowledgeBaseParams": {
 "BedrockKnowledgeBaseId": "SOME_ID",
 "OverrideSearchType": null
 }
 },
 "LlmParams": {
 "ModelProvider": "Bedrock",
 "BedrockLlmParams": { "ModelId": "anthropic.claude-v2" },
 "PromptParams": {
 "PromptTemplate": "some prompt",
 "MaxPromptTemplateLength": 187500,
 "MaxInputTextLength": 187500,
 "UserPromptEditingEnabled": true,
 "DisambiguationEnabled": true,
 "DisambiguationPromptTemplate": "some prompt"
 },
 "ModelParams": {},
 "Temperature": 1,
 "RAGEnabled": true,
 "Streaming": true,
 "Verbose": false
 }
}
```

Monitor the solution with Service Catalog AppRegistry

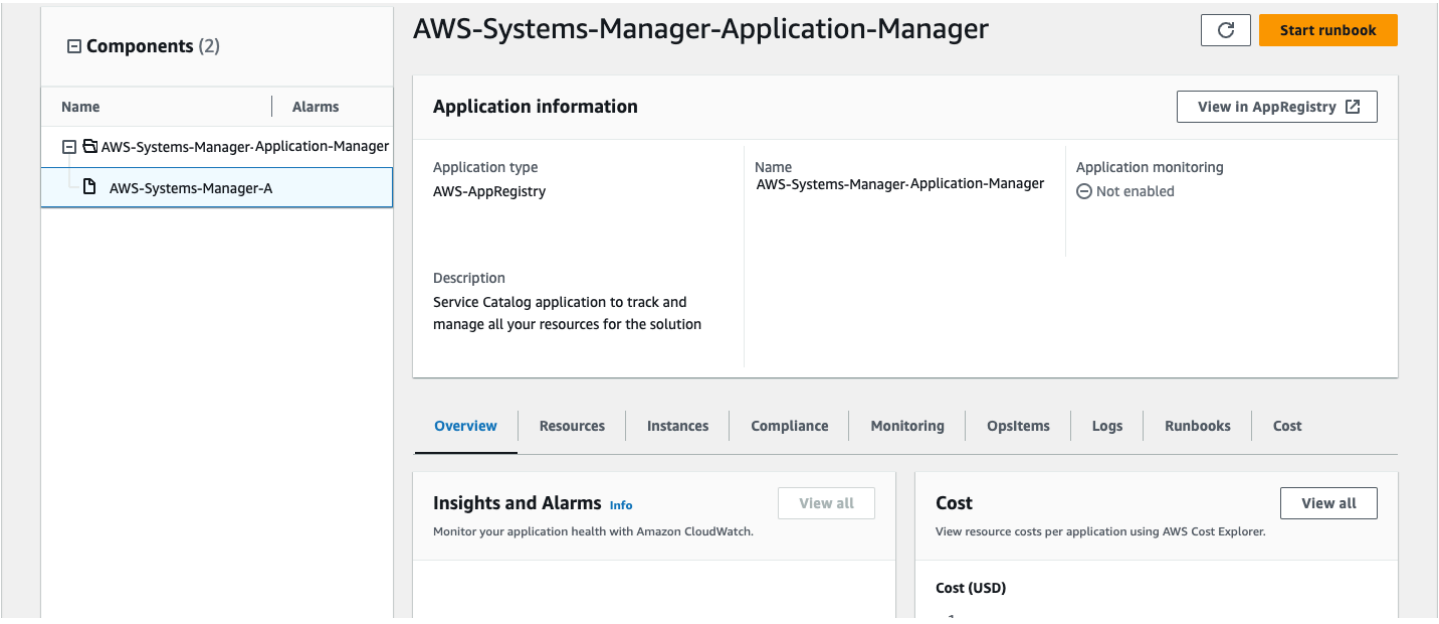
The solution includes a Service Catalog AppRegistry resource to register the CloudFormation template and underlying resources as an application in both Service Catalog AppRegistry and Systems Manager Application Manager.

Systems Manager Application Manager gives you an application-level view into this solution and its resources so that you can:

- Monitor its resources, costs for the deployed resources across stacks and AWS accounts, and logs associated with this solution from a central location.
- View operations data for the resources of this solution in the context of an application. For example, deployment status, CloudWatch alarms, resource configurations, and operational issues.

The following figure depicts an example of the application view for the solution stack in Application Manager.

Depicts solution stack in Application Manager



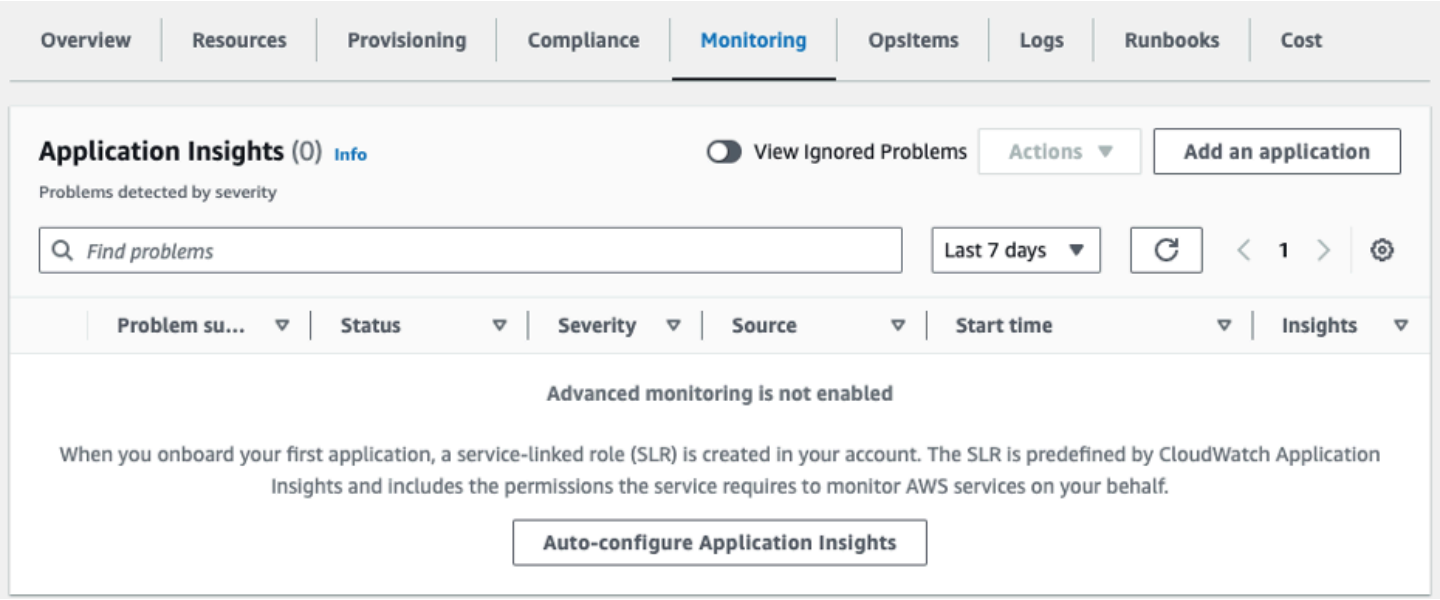
Activate CloudWatch Application Insights

1. Sign in to the Systems Manager console.

2. In the navigation pane, choose **Application Manager**.
3. In **Applications**, search for the application name for this solution and select it.

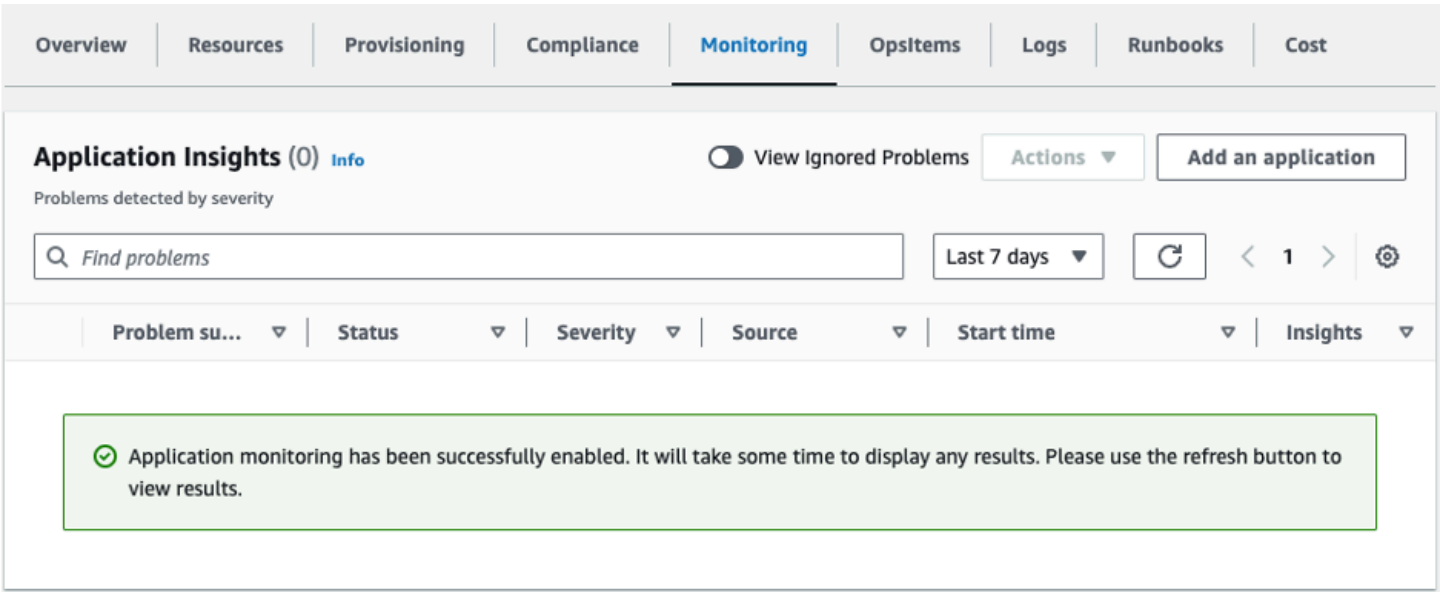
   The application name will have **App Registry** in the **Application Source** column, and will have a combination of the solution name, Region, account ID, or stack name.
4. In the **Components** tree, choose the application stack you want to activate.
5. In the **Monitoring** tab, in **Application Insights**, select **Auto-configure Application Insights**.

Application Insights dashboard showing no detected problems and option to auto-configure.



Monitoring for your applications is now activated and the following status box appears:

Application Insights dashboard showing successful monitoring activation message.



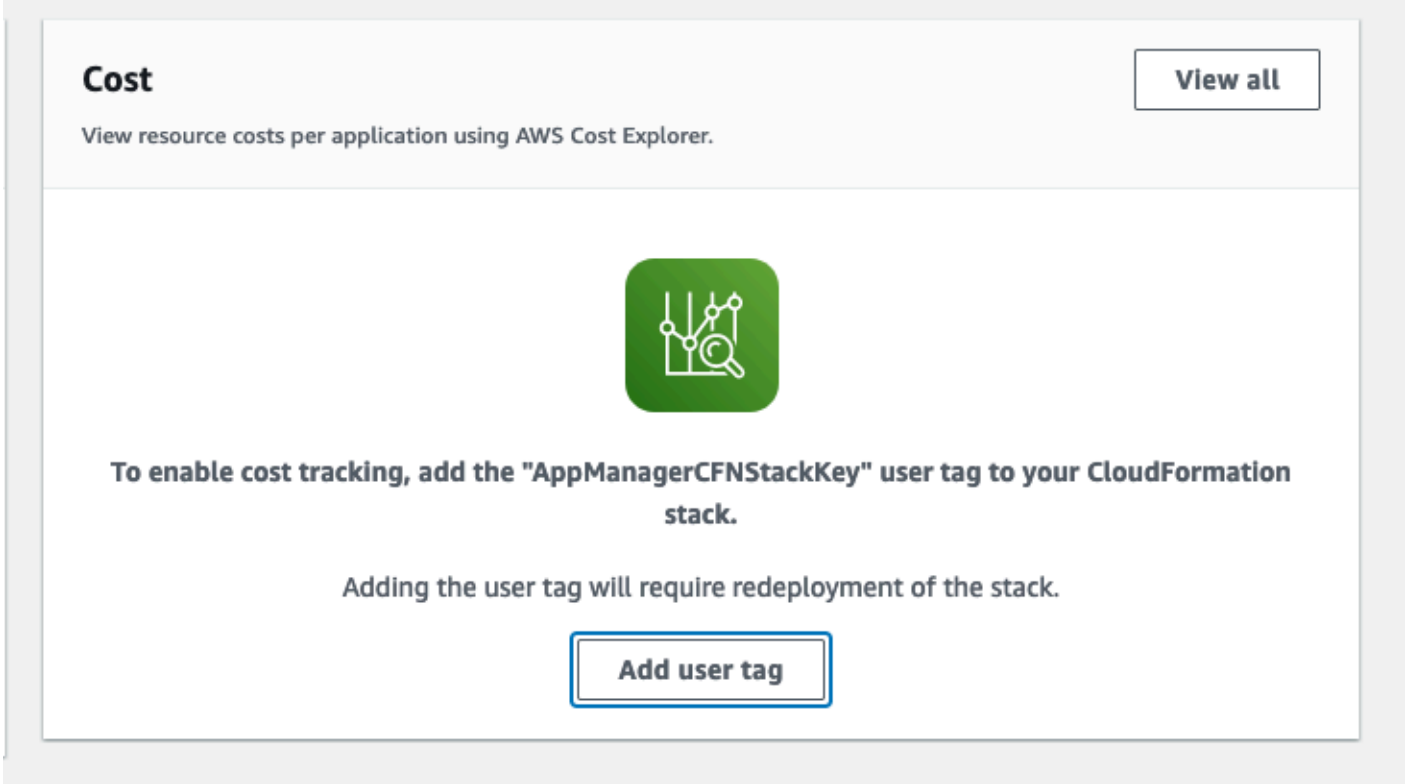
Confirm cost tags associated with the solution

After you activate cost allocation tags associated with the solution, you must confirm the cost allocation tags to see the costs for this solution. To confirm cost allocation tags:

1. Sign in to the Systems Manager console.
2. In the navigation pane, choose **Application Manager**.
3. In **Applications**, choose the application name for this solution and select it.

   The application name will have **App Registry** in the **Application Source** column, and will have a combination of the solution name, Region, account ID, or stack name.
4. In the **Overview** tab, in **Cost**, select **Add user tag**.

   Screenshot depicting the Application Cost add user tag screen



5. On the **Add user tag** page, enter `confirm`, then select **Add user tag**.

The activation process can take up to 24 hours to complete and the tag data to appear.

Activate cost allocation tags associated with the solution

After you activate Cost Explorer, you must activate the cost allocation tags associated with this solution to see the costs for this solution. The cost allocation tags can only be activated from the management account for the organization. To activate cost allocation tags:

1. Sign in to the AWS Billing and Cost Management and Cost Management console.
2. In the navigation pane, select **Cost Allocation Tags**.
3. On the **Cost allocation tags** page, filter for the AppManagerCFNStackKey tag, then select the tag from the results shown.
4. Choose **Activate**.

AWS Cost Explorer

You can see the overview of the costs associated with the application and application components within the Application Manager console through integration with AWS Cost Explorer, which must be first activated. Cost Explorer helps you manage costs by providing a view of your AWS resource costs and usage over time. To activate Cost Explorer for the solution:

1. Sign in to the AWS Cost Management console.
2. In the navigation pane, select **Cost Explorer** to view the solution's costs and usage over time.

Update the solution

If you have previously deployed the solution, follow this procedure to update the solution's CloudFormation stack to get the latest features and enhancements. There are three parts to the upgrade process:

- Step 1: Update Deployment dashboard
- Step 2: Migrate use case configurations
- Step 3: Update use cases

Note

1. In v2.0.0, integration with Anthropic and Hugging Face was deprecated in favor of Amazon Bedrock and Amazon SageMaker AI. You can deploy models available through Hugging Face through SageMaker JumpStart. Refer to Use Hugging Face with Amazon SageMaker AI for more details.
2. Ensure you test the update process in a non-production environment before running these steps.

Step 1: Update Deployment dashboard

1. Sign in to the CloudFormation console, select your existing CloudFormation stack, and select **Update**.
2. Select **Replace current template**.
3. Under Specify template:
   a. Select **Amazon S3 URL**.
   b. Copy the latest CloudFormation template link.
   c. Paste the link in the **Amazon S3 URL** box.
   d. Verify that the correct template URL shows in the **Amazon S3 URL** text box, and choose **Next**. Choose **Next** again.
4. Under **Parameters**, review the parameters for the template and modify them as necessary. For details about the parameters, see Step 1: Launch the Deployment dashboard stack.

5. Choose **Next**.
6. On the **Configure stack options** page, choose **Next**.
7. On the **Review** page, review and confirm the settings. Check the box acknowledging that the template will create IAM resources.
8. Choose **View change set** and verify the changes.
9. Choose **Update stack** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive an UPDATE_COMPLETE status in approximately 10 minutes.

If the existing Solution version was prior to v2.0.0, updating will create a web UI stack (which replaces the `amplify-ui` implementation of the login screen with a Cognito hosted UI) and a new CloudFront URL, which can be obtained from the Output section of the CloudFormation console once the stack status is UPDATE_COMPLETE.

> **Note**
>
> Existing use cases created using versions prior to v2.0.0 will NOT be displayed until you complete the steps outlined below.

Step 2: Migrate use case configurations (Only updates from versions below 2.0.0)

The schema for storing and the AWS service to store use case configuration has changed in version 2.0.0. Follow the steps described in GAAB v2 Migration User Guide using the gaab_v2_migration.py script. After you run the script, you can access the Deployment dashboard to view the deployed use cases.

> **Note**
>
> You must follow the steps below to complete migrating the use cases.

Step 3: Update use cases

You can edit the deployed use cases with new features available in the latest versions of GAAB. See Use the solution for information about how to use the features in this solution.

To update use cases to the latest version, you must complete the `Edit`use case steps in the Deployment dashboard (although you might not make any changes). This action triggers a CloudFormation stack update with the latest template version.

> **Note**
>
> Use cases created with 1.x or 2.x versions of the solution might not work with later versions. Hence, we recommend cloning existing use cases created with versions prior to v3.0.0 through the Deployment dashboard. Then, gradually migrate and replace with new use cases created using v3.0.0 or later.

Troubleshooting

This section provides troubleshooting instructions for deploying and using the solution.

If these instructions don't address your issue, Contact Support provides instructions for opening an Support case for this solution.

Problem: Deploying a VPC-enabled configuration, with Create a VPC for me, fails

The Deployment dashboard stack or the use case stack fails deployment because the CloudFormation was not able to provision VPC networking resources.

Resolution

Check the quota limits for VPCs, and Elastic IPs in your account. Default limits are 5 each for Elastic IPs and VPCs per AWS account, per AWS Region.

> **Note**
>
> When the solution creates a VPC, a single VPC-enabled deployment (Deployment Dashboard or Use Case) is a 2-AZ deployment with 1 public and 1 private subnet in each AZ, each public subnet deploys 1 NAT Gateway. With 2 NAT Gateways, the deployment consumes 2 public IP addresses from the quota limit.

Some limits to be aware of (per account, per Region):

- Number of VPCs - 5
- Number of public IP addresses - 5
- Number of Gateway VPC Endpoints - 20
- Number of Inteface VPC Endpoints - 20

Problem: Use case stack can't be deleted in CloudFormation after the Deployment dashboard stack gets deleted

If the Deployment dashboard stack is deleted in CloudFormation before all of the use case stacks are deleted, the use cases can end up in a locked (unusable) state. This is due to an IAM role created by the Deployment dashboard stack no longer exists preventing modifications to the use case stack.

Resolution

> **Warning**
>
> Ensure you clean up any manually created roles immediately after usage. These are elevated permissions that users could exploit for role elevation.

Recreate the deleted IAM role to enable the deletion of the CloudFormation stacks:

1. Open the CloudFormation console and determine the role that is associated with your locked stack.
   a. The role ARN can be found in the stack info section labeled **IAM role.**
   b. The role name is what follows after **:role/** in the IAM role ARN (for example, **arn:aws:iam::<account-id>:role/<role-name>**)
2. Create a new role in IAM with the same name as the deleted role.
   a. Select **AWS service** as the trusted entity and select **CloudFormation** from the drop down.
   b. Add the necessary permissions. If you're unsure about the required permissions, you can use the AWS managed **AdministratorAccess** policy.
   c. Enter the role name exactly as obtained in Step 1.
3. Return to the CloudFormation console and delete the locked stacks.
4. Once all locked stacks have been successfully deleted, return to IAM and delete any roles created in Step 2.

Problem: Use case UI does not reflect changes in settings

When use cases are updated, the UI is deployed to CloudFront. However, because CloudFront caches deployments as well as the configuration file that dictates how some settings are shown to the user, these changes might not be reflected immediately.

Resolution

The CloudFront distribution can be invalidated to force the new configuration to be propagated to frontend users.

1. Open the CloudFormation console and determine the CloudFront distribution that is associated with your use case stack.
   a. The use case stack should start with the same name you used when deploying the use case.
   b. Locate the nested stack corresponding to the UI. The nested stack name should begin with **WebAppS3UINestedStackS3UINestedStackResource**.
   c. Under the **Resources** tab, locate the resource of type **AWS::CloudFront::Distribution**, then select the physical ID. This will open the distribution in the CloudFront console.
2. Navigate to the **Invalidations** tab, then choose **Create Invalidation**, and input a path of /*. This will invalidate all paths.
3. In your own browser, delete any cookies and cached files related to the use case.

Contact AWS Support

If you have AWS Business Support+, AWS Enterprise Support, or Unified Operations, you can use the AWS Support Center to get expert assistance with this solution. The following sections provide instructions.

Create case

1. Sign in to Support Center.
2. Choose **Create case**.

How can we help?

1. Choose **Technical**.

2. For **Service**, select **Solutions**.
3. For **Category**, select **Other Solutions**.
4. For **Severity**, select the option that best matches your use case.
5. When you enter the **Service**, **Category**, and **Severity**, the interface populates links to common troubleshooting questions. If you can't resolve your question with these links, choose **Next step: Additional information**.

Additional information

1. For **Subject**, enter text summarizing your question or issue.
2. For **Description**, describe the issue in detail, including the name of this solution: **Generative AI Application Builder on AWS**.
3. Choose **Attach files**.
4. Attach the information that AWS Support needs to process the request.

Help us resolve your case faster

1. Enter the requested information.
2. Choose **Next step: Solve now or contact us**.

Solve now or contact us

1. Review the **Solve now** solutions.
2. If you can't resolve your issue with these solutions, choose **Contact us**, enter the requested information, and choose **Submit**.

Uninstall the solution

> **Note**
>
> Deployments created through the Deployment dashboard are not intended to be managed outside of the solution. Be sure to delete and clean up any deployments from within the Deployment dashboard, before deleting the stack in CloudFormation.

You can uninstall the Generative AI Application Builder on AWS solution from the AWS Management Console or by using the AWS Command Line Interface. You must manually delete the Amazon S3 buckets, Amazon Kendra indexes, or CloudWatch Logs created by this solution. AWS Solutions do not automatically delete Amazon S3 buckets, Amazon Kendra indexes, or CloudWatch Logs in case you have stored data to retain.

Using the AWS Management Console

1. Sign in to the AWS CloudFormation console.
2. On the **Stacks** page, select this solution's installation stack.
3. Choose **Delete**.

Using AWS Command Line Interface

Determine whether the AWS Command Line Interface (AWS CLI) is available in your environment. For installation instructions, see What Is the AWS Command Line Interface in the *AWS CLI User Guide*. After confirming that the AWS CLI is available, run the following command.

```
$ aws cloudformation delete-stack --stack-name <installation-stack-name>
```

Manual uninstall steps

Deleting the Amazon S3 buckets

This solution is configured to retain the solution-created Amazon S3 bucket if you decide to delete the AWS CloudFormation stack to prevent accidental data loss. After uninstalling the solution, you

can manually delete this Amazon S3 bucket if you do not need to retain the data. Follow these steps to delete the Amazon S3 bucket.

1. Sign in to the Amazon S3 console.
2. In the navigation pane, select **Buckets**.
3. Locate the *<stack-name>* S3 buckets.
4. Select the S3 bucket and choose **Delete**.

To delete the S3 bucket using AWS CLI, run the following command. You won't need to empty the bucket first when using the --force option.

```
$ aws s3 rb s3://<bucket-name> --force
```

Deleting the Amazon Kendra indexes

To prevent accidental data loss, this solution is configured to retain the solution-created Amazon Kendra indexes when the AWS CloudFormation stack has been deleted. After uninstalling the solution, you can manually delete the Amazon Kendra indexes that you no longer need to retain data for. Follow these steps to delete the Amazon Kendra index.

1. Sign in to the Amazon Kendra console.
2. In the navigation pane, select **Indexes**.
3. Locate and select the index you want to delete.
4. Choose **Delete** to delete the selected index.

To delete the Amazon Kendra index using AWS CLI, run the following command:

```
$ aws kendra delete-index --id<index-id>
```

Deleting the CloudWatch Logs

To prevent accidental data loss, we configured this solution to retain the CloudWatch Logs if you decide to delete the CloudFormation stack. After uninstalling the solution, you can manually delete the logs if you don't need to retain the data. Follow these steps to delete the CloudWatch Logs.

1. Sign in to the Amazon CloudWatch console.

2. In the navigation pane, select **Log Groups**.
3. Locate the log groups created by the solution.
4. Select one of the log groups.
5. Choose **Actions** and then choose **Delete**.

Repeat the steps until you have deleted all the solution log groups.

Use the solution

Accessing the UI

During the stack deployment process (for both the Deployment dashboard and use cases) an email is sent to the configured email address. The email contains the user's temporary credentials they can use to sign up and access the web interface.

> **Note**
>
> The DevOps user with access to the AWS Management Console must provide the admin user with the CloudFront URL of the Deployment dashboard UI when the stack completes.

For the use cases, the admin user with access to the Deployment dashboard UI must provide the business user with the CloudFront URL of the use case UI when the deployment completes.

Once logged in, the user can interact with the solution UIs, either the Deployment dashboard in the case of admins, or the use case in the case of business users.

How to update a deployment

When on the Deployment dashboard home page (or the details page of a deployment) you can edit the configuration used by a deployment. You can only edit deployments that are in the CREATE_COMPLETE or UPDATE_COMPLETE statuses.

Except for the use case name, all other options are editable for a deployment. Just change the values you want to edit and redeploy.

Depending on the scope of edits made, the redeployment time will vary. It might take a few seconds if simple settings have changed (example, model parameters), to more than 30 minutes if larger infrastructure related options have changed (example, request to create the Amazon Kendra index for the Text use case RAG).

Once the edit has completed successfully, the application status will report an UPDATE_COMPLETE status. At this time, you can access the deployed UI through the CloudFront URL and interact with the modified deployment.

Note

It might be easier to run multiple deployments side-by-side if you want to compare different settings or LLMs. Use the **Clone** feature to quickly use an existing configuration to launch a new deployment.

How to clone a deployment

When on the Deployments dashboard home page (or the details page of a deployment) you can clone the configuration used by a deployment. Cloning a deployment launches the **Deploy new use case** wizard, but with most fields pre-filled with the same values.

This is a convenience operation to help you quickly duplicate deployments with changed settings, revive a deleted deployment, or compare multiple LLMs in otherwise identical deployments.

How to delete a deployment

When on the Deployments dashboard home page (or the details page of a deployment) you can delete it once you no longer need the deployment. Deleting a deployment invokes a CloudFormation stack delete operation and deprovisions the resources for the deployment.

By default, a deleted deployment still remains on the dashboard to enable the clone functionality. To completely remove a deployment from the dashboard so that it stops being tracked in the UI, choose **Permanently delete** on the delete confirmation window.

Important

Some resources are left behind during stack deletion and must be manually deleted. Refer to the Manual uninstall section for details on what resources are retained and how to clean them up.

Configuring a Large Language Model (LLM)

Which LLM is right for your use case depends on a large set of factors specific to your needs and the type of customer experience you want to curate. This solution does not look to be prescriptive, but rather aims to give you the necessary tools to evaluate what works best for your application.

The AI-generated space is evolving rapidly, so it is incumbent on you to keep up to date on the latest models, optimization techniques, and best practices to ensure you are building the right experiences for your customers.

> **Note**
>
> If you're working with non-public or sensitive data, then be sure to select an LLM option using AWS services (such as Amazon Bedrock or Amazon SageMaker AI). This improves the overall security posture of your deployment by keeping data within your Region and on the AWS network when compared to using an LLM hosted by a third-party provider.

Using Amazon SageMaker AI as an LLM Provider

As of v1.3.0, Amazon SageMaker AI is available as a model provider for Text use cases. This feature allows you to use a SageMaker AI inference endpoint already existing within the AWS account in the solution. Here are some ways to get started.

> **Important**
>
> The solution does not manage the lifecycle of your SageMaker AI endpoints. You are responsible for deleting the SageMaker AI endpoints once they are no longer needed to stop incurring additional charges.

Creating a SageMaker AI endpoint

You can use Amazon SageMaker AI JumpStart to quickly deploy an endpoint.

You can also use a text-generation based SageMaker AI endpoint and deploy using the base SageMaker AI service. Refer to the SageMaker AI JumpStart documentation for a step by step guide on how to deploy a model for inference.

> **Note**
>
> Foundation models/LLMs are typically quite large and can often require the use of large accelerated compute instances. Many of these larger instances might not be available by

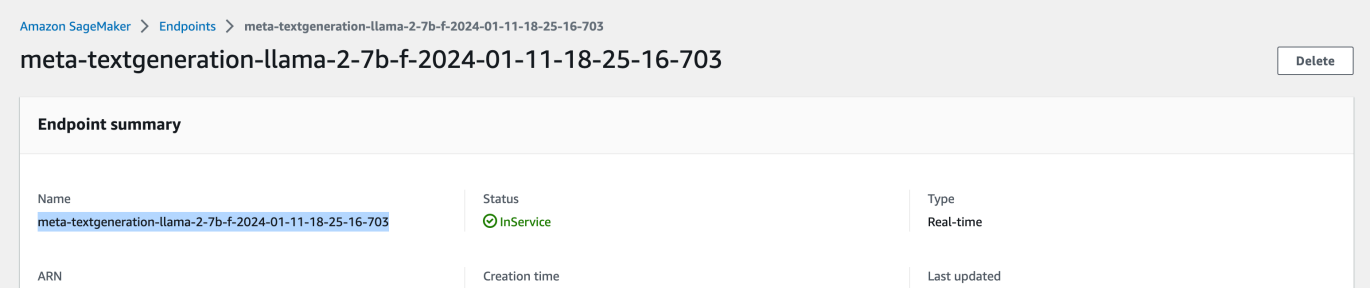
default in your AWS account. Refer to the default SageMaker AI quotas and be sure to request a quota increase before deploying to avoid common deployment failures.

Use SageMaker AI endpoint to create a Text use case deployment

To deploy a new Text use case using a SageMaker AI endpoint for inference:

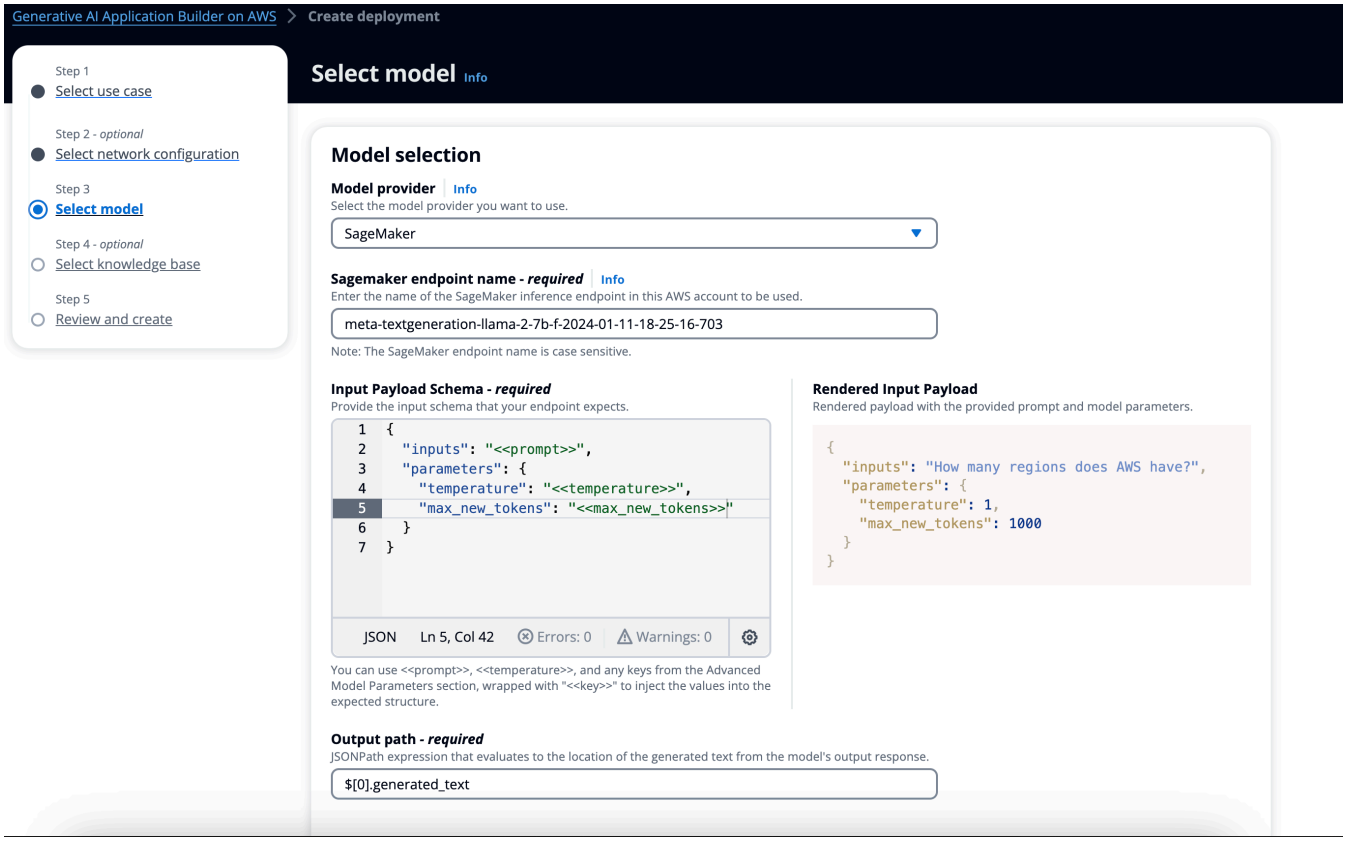
1. Create a new use case through the Deployment dashboard wizard and complete the forms until you reach the Models selection page.
2. On the Models page, select **SageMaker AI** as the model provider. This will generate a custom form requiring three key pieces of user input:
   - The name of the SageMaker AI endpoint you want to use. DevOps users can obtain this from the AWS console. Note that the endpoint must be in the same account and Region as the solution is deployed in.

     Location of the endpoint name on the AWS console

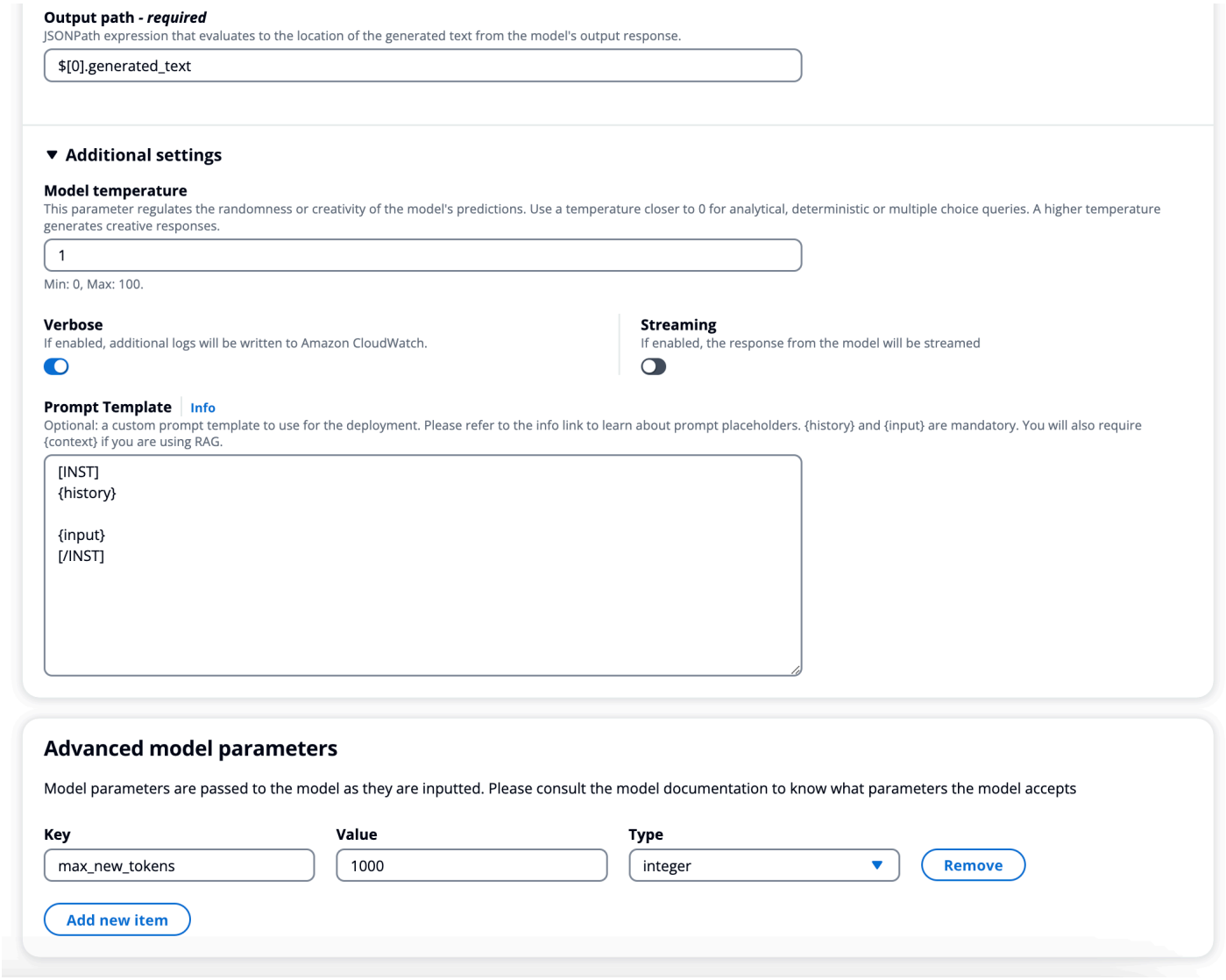
   - The schema of the input payload expected by the endpoint. To support the widest set of endpoints, admin users are required to tell the solution how their endpoint expects the input to be formatted. In the model selection wizard, provide the JSON schema for the solution to send to the endpoint. You can add placeholders to inject static and dynamic values into the request payload. The available options are:
     - Mandatory placeholders: \<\<prompt\>\> will be dynamically replaced with the full input (for example, history, context, and user input as per the prompt template) to be sent to the SageMaker AI endpoint at runtime.
     - Optional placeholders: \<\<temperature\>\> *,* as well as any parameters defined in advanced model parameters can be provided to the endpoint. Any string containing a placeholder enclosed in \<\< and \>\> (for example, \<\<max_new_tokens\>\>) will be replaced by the value of the advanced model parameter of the same name.

Example input schema - setting mandatory fields, prompt and temperature, along with a custom advanced parameter, max_new_tokens. Output path must be supplied as a valid JSONPath string



3. The location of the LLMs generated string response within the output payload. This must be supplied as a JSONPath expression to indicate where the final text response shown to users is expected to be accessed from within the endpoint's return object and response.

   Example of adding Advanced model parameters to use within SageMaker AI input schema (see Figure 2 for previous options/settings)



Note

SageMaker AI now supports hosting multiple models behind the same endpoint, and this is the default configuration when deploying an endpoint in the current version of SageMaker AI Studio (not Studio Classic).

If your endpoint is configured in this way, you will be required to add **InferenceComponentName** to the advanced model parameters section, with a value corresponding to the name of the model you want to use.

Advanced LLM Settings

While using Amazon Bedrock, you can configure some advanced settings for your models such as Amazon Bedrock Guardrails, Provisioned Throughput for Amazon Bedrock, and additional model parameters.

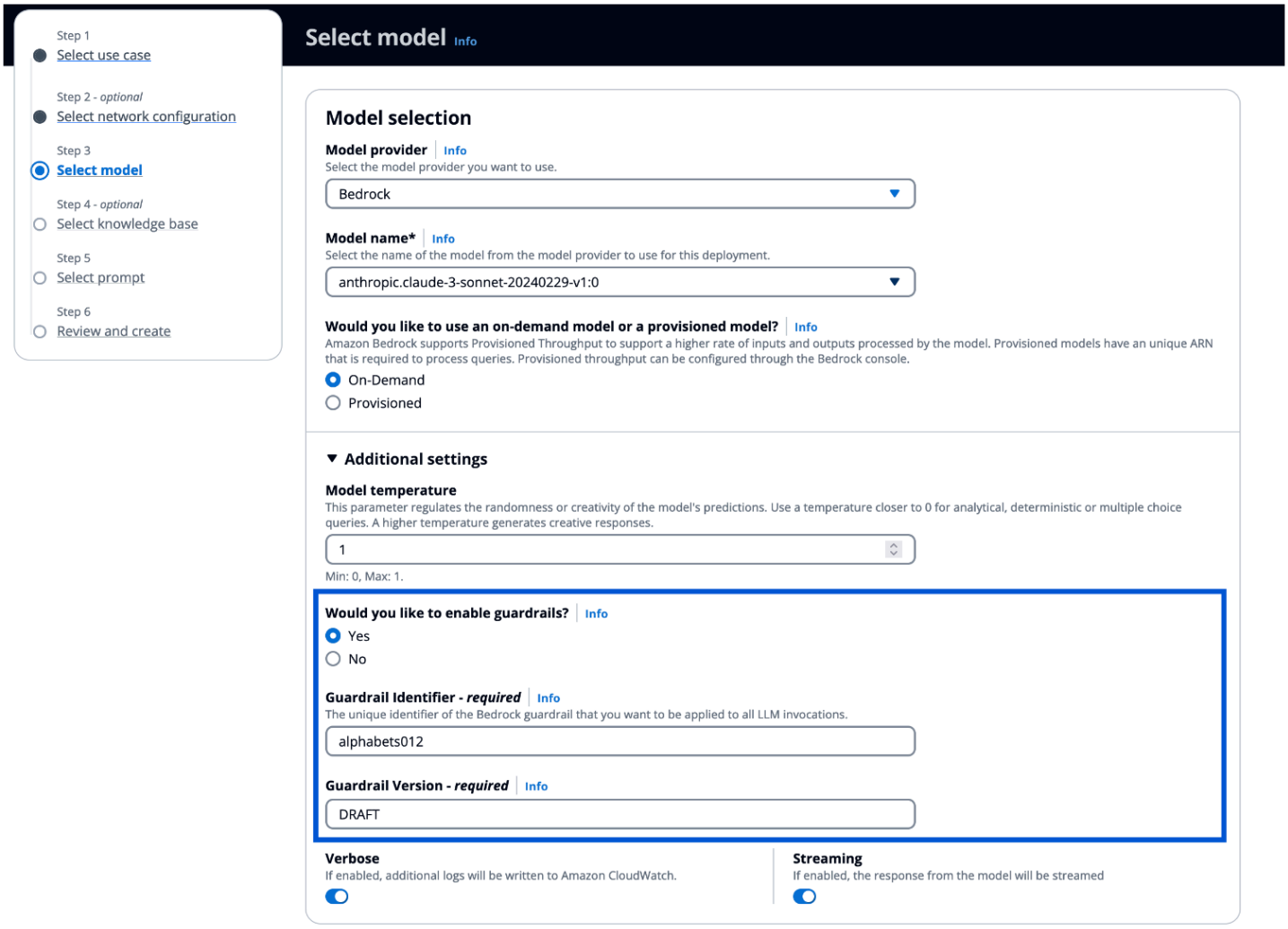
Amazon Bedrock Guardrails

Amazon Bedrock Guardrails is a feature with Amazon Bedrock which evaluates user inputs and LLM responses based on user configured policies and provides an additional layer of safeguards, regardless of the underlying LLM that the user selects for a use case. A Guardrail consists of 2 policies to avoid content that falls into undesirable or harmful categories:

1. Denied topics to define a set of topics that are undesirable in the context of user's application, for example, investment advice in a financial application, and,
2. Content filters****which allows filtering input user prompts or model responses containing harmful content.

For usage in Generative AI Application Builder solution, a Guardrail must be configured in the *Amazon Bedrock* console using the *Create guardrail* wizard. Once created, you can add this Guardrail to your chat use case created through Generative AI Application Builder solution wizard in the **Additional settings** in the Model Selection step by supplying your Guardrail Identifier and Guardrail version.

Depicts Deployment wizard - enabling Amazon Bedrock Guardrails



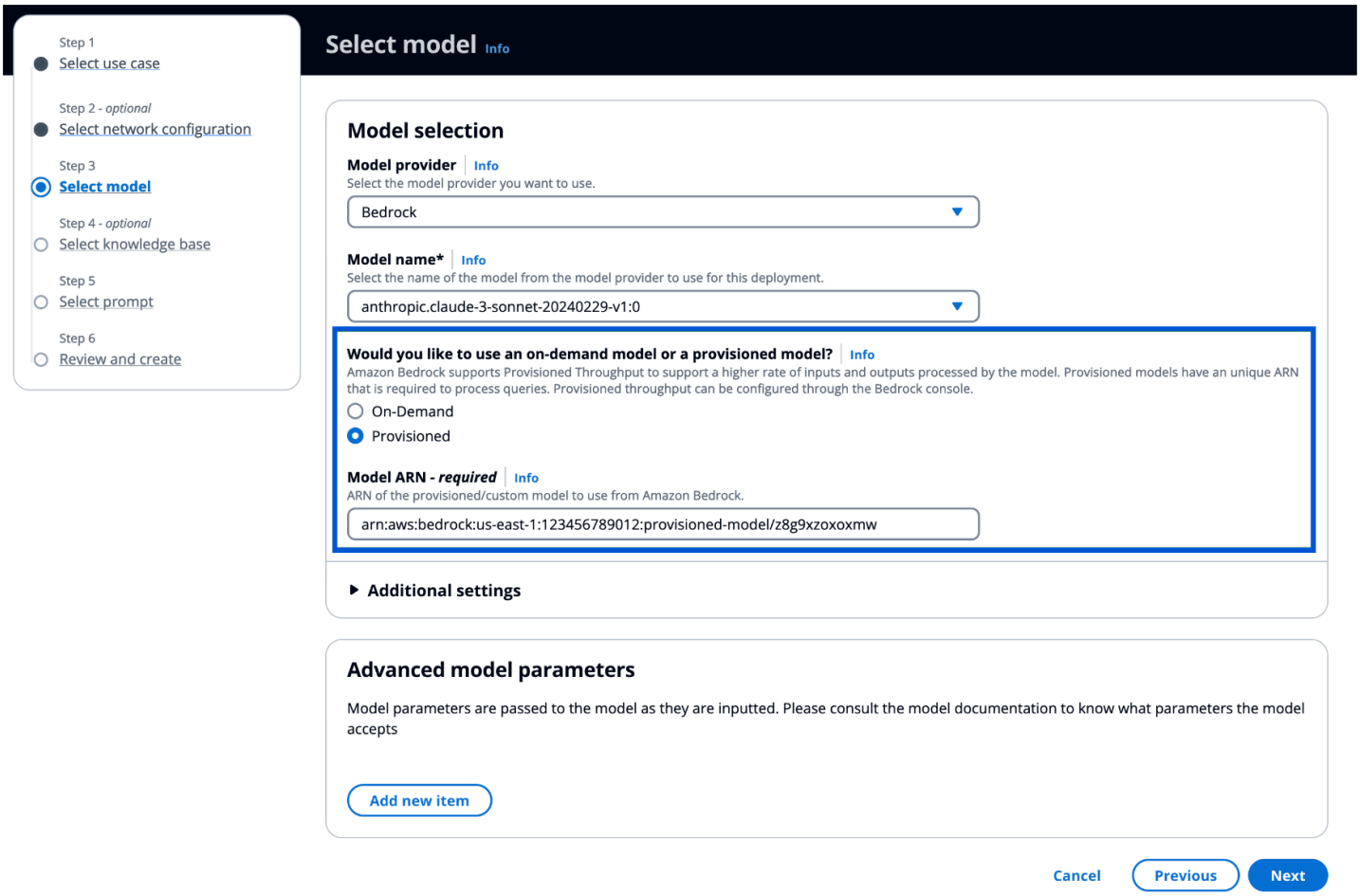
Provisioned Throughput for Amazon Bedrock

Each on-demand Amazon Bedrock model follows region-specific account quota limit for model inferencing. For example, Anthropic Claude 2.x on Bedrock currently allows for 500 requests and 500,000 tokens processed per minute in us-east-1 and us-west-2 regions. You may also want to use the solution with your fine-tuned or continued pre-trained models. For such instances, Amazon Bedrock allows provisioned throughput which allows running large consistent inference workloads for your base, fine-tuned or continued pre-trained models for use in production-grade applications.

Once Provisioned Throughput is purchased within the Amazon Bedrock console, a Model ARN is generated for usage. You can now supply this Model ARN in the Generative AI Application Builder wizard in the Model selection step. To do so, select Bedrock as the model provider and the base model name which was used to generate this provisioned Model ARN in Amazon Bedrock console.

Then, select '**Provisioned model**' when choosing between on-demand and provisioned models, and supply your Model ARN.

Depicts Deployment wizard - Enabling Provisioned Throughput for Amazon Bedrock



> **Note**
>
> Your guardrail and provisioned throughput must be in the same Region as the deployed Deployment Dashboard and use case stacks.

Model parameters

LLMs often accept a wide range of parameters specific to its implementation. Model providers often provide documentation outlining the set of supported parameters and their uses.

The solution passes model parameters directly through to the underlying model so it is important to ensure parameters are set correctly. Refer to the model provider's documentation for the latest information on supported parameters.

Configuring Agent Builder

Agent Builder provides comprehensive configuration options for creating production-ready AI agents. This section describes how to configure and manage Agent Builder deployments.

System prompt configuration

The system prompt defines your agent's behavior, personality, and capabilities. To configure the system prompt:

1. In the Agent Builder wizard, navigate to the **Configure Agent** step.
2. Edit the system prompt template in the text editor.
3. Include clear instructions for:
   - Agent's role and purpose
   - How to use available tools (MCP servers)
   - Response formatting preferences
   - Behavioral guidelines
4. Use the **Reset to default** button to restore the original template if needed.

Best practices for agent prompts:

- Be specific about the agent's capabilities and limitations
- Provide clear examples of desired behavior
- Include instructions for tool usage and when to invoke them
- Define response format expectations
- Set boundaries for agent behavior

MCP server integration

Model Context Protocol (MCP) servers provide agents with access to enterprise tools and data sources. To configure MCP servers:

1. In the **Configure Agent** step, locate the **MCP Servers** section.
2. Select from available MCP servers in the dropdown menu.

> **Note**
>
> MCP servers must be configured and accessible before agent deployment. The agent will automatically discover and use tools exposed by the configured MCP servers. Refer to the MCP documentation for server setup and tool configuration.

Memory settings

Agent Builder provides two types of memory for maintaining context and knowledge:

Short-term memory

Enabled by default for all agents:

- Maintains conversation context within sessions
- Automatically captures user messages and agent responses
- Organized by actorId and sessionId for proper isolation
- No configuration required

Long-term memory

Optional feature for storing insights across sessions:

1. In the **Configure Agent** step, locate the **Memory Configuration** section.
2. Toggle **Enable long-term memory** to activate.
3. When enabled, the agent can:
   - Extract and store important information across conversations
   - Retrieve relevant context from previous sessions
   - Build knowledge about user preferences and history

Note

Long-term memory uses AgentCore Memory with semantic memory strategy and default retention settings.

Monitoring Agent Builder deployments

Agent Builder provides comprehensive monitoring through CloudWatch dashboards and metrics.

Accessing CloudWatch dashboards

1. Navigate to the CloudWatch console in your AWS account.
2. Select **Dashboards** from the left navigation.
3. Find the dashboard named `AgentBuilder-<UseCaseId>`.
4. View real-time metrics and historical performance data.

Log access and analysis

Agent logs are available in CloudWatch Logs:

1. Navigate to CloudWatch Logs in the AWS console.
2. Find log groups prefixed with `/aws/bedrock-agentcore/runtimes/`.
3. Use CloudWatch Insights to query and analyze logs.
4. Search for specific request IDs or error patterns.

Configuring Workflow Builder

Workflow Builder enables multi-agent orchestration through a supervisor agent that delegates work to specialized Agent Builder agents.

Creating a workflow

1. Navigate to the Deployment Dashboard
2. Select **Create Workflow Use Case**

3. Configure the supervisor agent:
   - **Name**: Descriptive name for the workflow
   - **Description**: Purpose and capabilities
   - **System Prompt**: Instructions for agent delegation and coordination
   - **Model**: Foundation model for the supervisor agent

Best practices for supervisor prompts:

- Clearly describe when to use each specialized agent
- Include instructions for aggregating results from multiple agents
- Define response formatting expectations
- Set boundaries for delegation behavior

Agent selection

Select Agent Builder agents to include as specialized agents:

1. Click **Add Agent** in the workflow configuration
2. Browse or search available Agent Builder agents
3. Review agent descriptions
4. Select agents to include in the workflow

Agent descriptions

The supervisor agent uses agent descriptions to decide which agent to delegate to. Ensure descriptions clearly explain:

- Agent's specialized domain or capability
- Types of tasks the agent handles
- Input/output expectations

Testing workflows

After deployment:

1. Access the workflow through the Deployment Dashboard
2. Test with queries that require multiple agents
3. Monitor agent delegation in CloudWatch logs
4. Review response quality and delegation patterns
5. Adjust supervisor prompt if delegation is suboptimal

Tips for managing model token limits

Note: The solution does not directly attempt to manage token limits imposed by various LLMs. Test and ensure your prompt remains within the available limits enforced by the model provider.

To help control the size of prompts, try the following:

1. Familiarize yourself with the limits imposed by the model you want to use. These values can differ dramatically across models so it's important to know what your available budget is before getting started.
2. Craft your initial prompt with that budget in mind and consider how much you want to save for any dynamic elements of the prompt. For example, user input, chat history, document excerpts, and so on.
3. In the prompt configuration page, set a limit for **Size of trailing history** to limit the number of conversation turns included within the prompt.
4. Set document return limits in the Knowledge Base configuration wizard. You need to try and strike the right balance between providing the LLM with enough context to perform the task, but not so much as to exceed token limits or negatively affect latency.
5. Leave some buffer. Don't budget for the typical case, think about and experiment with the edge cases such as long input queries, large document excerpts, or long conversations.

Steps to build MCP server Docker Image

To use MCP (Model Context Protocol) servers with Generative AI Application Builder on AWS, you need a Docker image built and stored in a private Amazon ECR repository as the first step.

Note

As of now, existing deployed MCP servers in Amazon Bedrock AgentCore runtime cannot be exported into GAAB. For MCP servers to be attached to Agents created through GAAB, they need to be created through GAAB.

Step 1: Create your MCP server

First, you need to have your MCP server implementation ready. For detailed instructions on creating an MCP server, refer to the Amazon Bedrock AgentCore Developer Guide - Create an MCP server.

We recommend the following project structure:

```
.
### __init__.py
### extras/
#   ### extra_dependencies.py
#   ### Dockerfile
### requirements.txt
### server.py <-- Server Entry point
```

For the Dockerfile structure, we recommend using a format similar to the following example:

```
FROM ghcr.io/astral-sh/uv:python3.13-bookworm-slim
WORKDIR /app

# All environment variables in one layer
ENV UV_SYSTEM_PYTHON=1 \
    UV_COMPILE_BYTECODE=1 \
    UV_NO_PROGRESS=1 \
    PYTHONUNBUFFERED=1 \
    DOCKER_CONTAINER=1 \
    AWS_REGION=us-east-1 \
    AWS_DEFAULT_REGION=us-east-1

COPY requirements.txt requirements.txt
# Install from requirements file
RUN uv pip install -r requirements.txt
```

```
RUN uv pip install aws-opentelemetry-distro>=0.10.1

# Signal that this is running in Docker for host binding logic
ENV DOCKER_CONTAINER=1

# Create non-root user
RUN useradd -m -u 1000 bedrock_agentcore
USER bedrock_agentcore

EXPOSE 9000
EXPOSE 8000
EXPOSE 8080

# Copy entire project (respecting .dockerignore)
COPY . .

# Use the full module path
CMD ["opentelemetry-instrument", "python", "-m", "server"]
```

Step 2: Test your MCP server locally

Before deploying to AWS, it's important to test your MCP server locally to ensure it works as expected. For detailed instructions on local testing, refer to the Amazon Bedrock AgentCore Developer Guide - Test your MCP server locally.

Step 3: Deploy to Amazon ECR

Once your MCP server is created and tested locally, follow these steps to deploy it to Amazon ECR:

1. Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see Getting Started with Amazon ECR.
2. Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS CLI:

   ```
   aws ecr get-login-password --region us-east-1 | docker login --username AWS --
   password-stdin <account-id>.dkr.ecr.us-east-1.amazonaws.com
   ```

3. Build your Docker image using the following command. For information on building a Docker file from scratch, see the Docker documentation. You can skip this step if your image is already built:

   ```
   docker build -t <repository-name> .
   ```

4. After the build completes, tag your image so you can push the image to this repository:

```
docker tag <repository-name>:latest <account-id>.dkr.ecr.us-east-1.amazonaws.com/
<repository-name>:latest
```

5. Run the following command to push this image to your newly created AWS repository:

```
docker push <account-id>.dkr.ecr.us-east-1.amazonaws.com/<repository-name>:latest
```

For complete deployment instructions, refer to the Amazon Bedrock AgentCore Developer Guide - Deploy your MCP server to AWS.

Step 4: Use the ECR URI in GAAB

After successfully pushing your Docker image to Amazon ECR, copy the image URI from the ECR console. You will use this URI when deploying your MCP server through the Generative AI Application Builder on AWS deployment wizard.

Steps to create different MCP Gateway Targets

Amazon Bedrock AgentCore Gateway allows you to transform existing AWS services and APIs into MCP tools that can be used by your agents. The Gateway supports multiple target types, enabling you to integrate various backend services seamlessly.

The following target types are supported:

- **Lambda targets**: Transform AWS Lambda functions into MCP tools. For detailed instructions, refer to the Amazon Bedrock AgentCore Developer Guide - Add Lambda targets.
- **OpenAPI targets**: Use OpenAPI specifications to define and expose REST APIs as MCP tools. For detailed instructions, refer to the Amazon Bedrock AgentCore Developer Guide - OpenAPI schema.
- **Smithy targets**: Build MCP tools using Smithy model definitions for type-safe API integrations. For detailed instructions, refer to the Amazon Bedrock AgentCore Developer Guide - Building Smithy targets.
- **MCP Server targets**: Connect directly to external MCP servers via URL endpoints, allowing you to integrate existing MCP servers. For detailed instructions, refer to the Amazon Bedrock AgentCore Developer Guide - MCP servers targets.

For additional examples and tutorials on creating MCP Gateway targets, visit the Amazon Bedrock AgentCore samples repository.

Configuring a knowledge base

This section describes how to ingest data into the knowledge base you've selected for the solution. The solution currently supports Amazon Kendra and Amazon Bedrock Knowledge Bases as knowledge bases for your RAG-based use case deployment.

Amazon Kendra

If you're using Amazon Kendra as your knowledge base, refer to the Amazon Kendra Developer Guide for information on how to use various data source connectors to help you ingest data from a wide selection sources.

Important: To prevent accidental data loss, the solution does not automatically delete the Kendra index (whether created by the solution or otherwise) when a deployment or stack is deleted. If you want to delete your knowledge base and stop incurring costs, see the Manual uninstall section for details on which resources are retained and how to clean them up.

Amazon Bedrock Knowledge Bases

Amazon Bedrock Knowledge Bases can be backed by a variety of different vector stores, each with the capability of indexing your data. To set up and populate your knowledge base, consult the Amazon Bedrock User Guide. Specifically, you will want to:

- First set up your data source
- Then set up a vector index for your knowledge base in a supported vector store. Note that this can be skipped if you use the "Quick create a new vector store" option in Bedrock console during knowledge base creation.
- Finally, you can create the knowledge base and sync your configured data sources.

Advanced knowledge base settings

Advanced Knowledge Base Settings such as Knowledge Base Filtering and RAG with Role Based Access Control are available for use with the solution. Knowledge Base Filtering can apply to either of the Knowledge Bases while RAG with Role Based Access Control is specifically available for Amazon Kendra.

Knowledge base filtering

The solution allows you to specify Amazon Kendra attribute filters or Bedrock knowledge base retrieval filters when deploying a use case in the Advanced RAG configurations section of the wizards knowledge base step. These filters define how data sources in the knowledge base are queried, such as search strategies, languages of the underlying document being queries, etc.

In both cases, a JSON object is used to specify the filter settings per the format specified in each services documentation (as linked above).

Example 1: Kendra AttributeFilter

```
{
 "EqualsTo": {
 "Key": "_language_code",
 "Value": {
 "StringValue": "es"
 }
 }
}
```

Example 2: Bedrock RetrievalFilter

```
{
 "equals": {
 "key": "language",
 "value": "es"
 }
}
```

RAG with Role Based Access Control with Amazon Kendra

Role-based access control (RBAC) allows controlling which users or groups can access certain documents in your Amazon Kendra index or see certain documents in their search results. To configure RBAC for your Amazon Kendra Index ID with your Generative AI Application Builder on AWS (GAAB) use case, follow these steps:

1. Configure Amazon Kendra Index

1. Ensure that you have an Amazon Kendra index created and at least one data source added to it.

2. Configure access control for your data source based on user groups. For an S3 data source, follow the instructions available in the documentation to set up access control lists (ACLs) using the same group names created in your Amazon Cognito User Pool. This ensures that users can only access the documents and search results they are authorized to view based on their group membership.

> **Note**
>
> Under **User Access Control** in the Kendra Index you created, leave **Token-based user access control** as **No**. When you enable **Role Based Access Control** in Step 2, Generative AI Application Builder on AWS extracts the appropriate claims from the user authentication token and creates an Attribute Filter.

2. Deploy RAG Use Case using GAAB Deployment Wizard

1. Follow the on-screen wizard instructions in the GAAB Deployment Wizard until you reach step 4 of the wizard to configure RAG.
2. In the **Select Knowledge Base** step of the deployment wizard, choose **Amazon Kendra** as the knowledge base type.
3. Specify whether you have an existing Amazon Kendra index or if you want to create a new one. If you have an existing index, provide the ID of your Amazon Kendra index that has been configured with access control lists (ACLs) based on user groups.
4. Enable the **Role Based Access Control** option. This option ensures that the search results returned from the Amazon Kendra index are filtered based on the user's role and group permissions.
5. Review and deploy the use case.

3. Configure Amazon Cognito

1. Locate the Amazon Cognito User Pool used by your GAAB deployment. This Amazon Cognito User Pool is typically created by the main deployment dashboard CloudFormation stack.
2. Create new users in the Amazon Cognito User Pool. When creating users, select the 'Send an email invitation' option so that users receive temporary login credentials via email. This allows new users to sign up and access the GAAB application.

3. Create user groups in the Amazon Cognito User Pool. Ensure that the group names exactly match the groups configured in your Amazon Kendra index ACLs. This is crucial for enabling RBAC, as the user's group membership will determine the search results they can access.
4. Assign users to the appropriate groups based on their roles and access permissions. Users must be added to both the group required for the Amazon Kendra index ACL, as well as the use case-specific group created during the GAAB deployment. This ensures that users have the necessary permissions to access the specific use case and the relevant search results.

By following these steps, you will have configured role-based access control (RBAC) for your GAAB deployment, ensuring that users can only access and interact with the information and features they are authorized for, based on their assigned user group and permissions.

> **Note**
>
> As of now, only Amazon Kendra supports RBAC for knowledge bases in the Generative AI Application Builder on AWS. For Amazon Bedrock Knowledge Base, RBAC is not supported, but you can use metadata filters to achieve some level of filtering. For more information, refer to the Amazon Bedrock User Guide.

Configuring your prompts

The Deployment dashboard wizard has a prompt configuration step which allows you to customize the prompt experience and template that will guide the interactions between users and the AI model. Properly configuring these settings is crucial for obtaining accurate and relevant responses from the AI assistant.

This section controls the overall experience and behavior of the AI prompt.

- **Max prompt template length**: This setting determines the maximum length (in characters) of the prompt template. A higher value allows for more context to be provided to the AI model, potentially leading to more accurate responses. However, excessively long prompts may also introduce noise and negatively impact performance. For Amazon Bedrock models, the default values for max prompt template length (in characters) is calculated using the underlying model token limits. If you edit and change a model name within Bedrock, 'Reset to default' button is highlighted and can be used to adopt the newly selected model's defaults. For Amazon SageMaker AI models, reasonable default values are provided, but it is recommended that

you check your underlying model and choose these max prompt template length and input text lengths accordingly. Refer to the Tips on managing model token limits section for more information.

- **Max input text length**: This setting limits the maximum length (in characters) of the user's input text. Longer inputs may contain irrelevant information, increasing the risk of obtaining irrelevant or inaccurate responses from the AI model.
- **User Prompt Editing**: This option allows you to enable or disable the ability for users to modify the prompt template through the Chat UI. Disabling this feature can help maintain consistency and prevent unintended changes to the prompt.

Prompt template

This section allows you to define the actual prompt template that will be used by the AI model. The prompt template typically follows a structure that includes placeholders for various components, such as the user's input, reference passages, and chat history.

- **Prompt template**: This is the main text area where you can write or paste the desired prompt template. The template should be crafted to provide the necessary context and instructions to the AI model. It typically includes the following placeholders:
  - `{input}`: This placeholder is mandatory for Sagemaker AI deployments and will be substituted with the user's input or query.
  - `{history}`: This placeholder is mandatory for Sagemaker AI deployments and will be substituted with the chat history of the current conversation.
  - `{context}`: This placeholder is mandatory for RAG deployments and will be substituted with the document excerpts obtained from the configured knowledge base.
- **Rephrase Question?**: This option (available for RAG deployments only) determines whether the user's original input query should be rephrased or disambiguated before being passed to the AI model. Rephrasing the query can sometimes help the model better understand the user's intent, potentially leading to more accurate responses.

When configuring the prompt template and experience, it's essential to strike a balance between providing sufficient context and instructions to the AI model while avoiding excessively long or irrelevant information that may introduce noise or performance issues.

Advanced prompt settings

This section allows you to control how the conversation history is presented to the AI model.

- **Size of trailing history**: This setting determines the number of previous messages that should be included in the final prompt. Setting this value to zero would result in no history being injected into either the prompt template or the disambiguation prompt template. Please note: even when set to zero, a {history} placeholder is still required to exist in the prompt templates. At runtime, it will get replaced with an empty string.
  - Note: It is recommended to provide an even number for this value. Providing an odd number would result in only the AI response of a paired interaction being returned.
- **Human Prefix**: This is the prefix used to identify messages sent by the user in the conversation history.
- **AI Prefix**: This is the prefix used to identify messages returned by the AI model in the conversation history.

Disambiguation Prompt Configuration

This section allows you to configure the behavior and template for disambiguating user inputs before sending them to the configured knowledge base.

- **Enable Disambiguation**: This option determines whether user inputs should be disambiguated before sending to the knowledge base.
- **Disambiguation Prompt Template**: This is the prompt template used for disambiguating user inputs when connected to a knowledge base. The output generated from this prompt will be used as the query sent to the knowledge base. Disabling disambiguation would result in the user's raw query being sent to the knowledge base unchanged.

For example, with disambiguation enabled, a follow-up user query of "How much does it cost?" might be disambiguated to "How much does it cost renew my license plate?", leading to a better search query.

Using the deployed Text use case

The built-in UI for the Text use case is intended to enable business users to quickly explore and experiment with the deployment created by the admin user. Configuration changes made by the business user only take effect for their session. The business user must share these changes with the admin user who can update the base deployment with those changes for all to use.

The chat UI consists of the following components:

- Chat window
- Chat input box
- Settings
- Clear conversation

Chat window

Holds different turns of the conversation. Messages starting on the right are from the business user, and messages starting on the left are from the configured LLM. A small clipboard icon exists on all LLM responses to enable easy copying of responses.

Chat input box

Pinned to the bottom of the chat window is the chat input box. This is where business users can enter their messages to be sent to the LLM. Just above the input box is the connection status. If the connection is lost (for example, due to inactivity), a new connection is automatically created the next time a chat message is sent. This request is expected to take a little longer due to the additional WebSocket connection time.

Based on the specific configuration, there might be a maximum length enforced on the input. If this limit is exceeded, users receive an alert and the message is not sent.

Note: If using RAG with Amazon Kendra, the Retrieve API will truncate queries to 30 token words. If expecting longer user inputs, evaluate how this might affect search performance.

Settings

To enable business users to quickly experiment with different configurations, a settings panel is available, which enables on-the-fly editing of certain deployment configuration options

(example, prompt template). These changes can only be made at the start of a new session. Once a conversation is started, clearing the conversation re-enables the editing of the configuration settings.

Note: Admin users can choose to lock a deployment's settings. They can prevent live edits at deployment time through the wizard during the prompt step.

Clear conversation

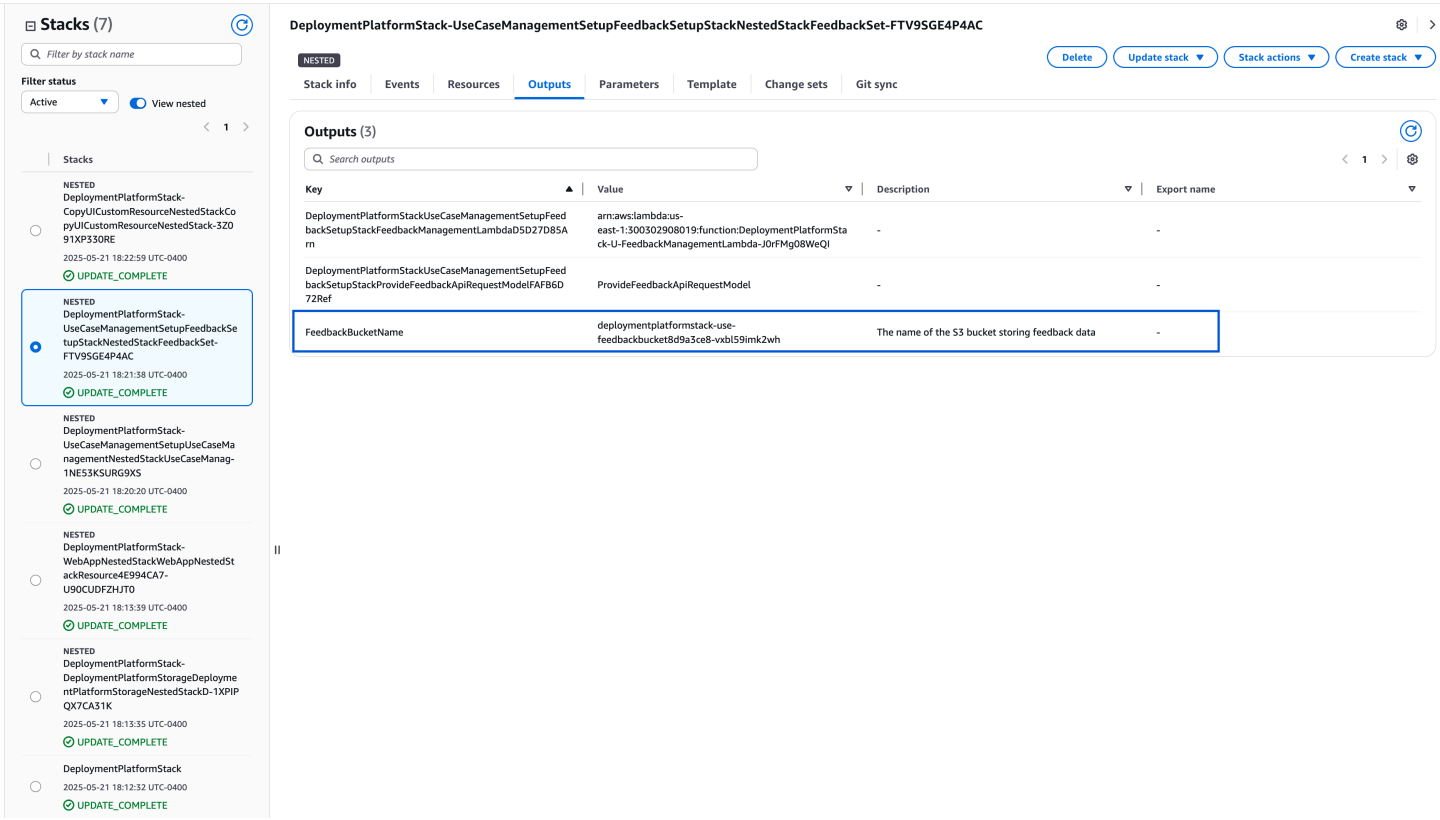
Over the course of the conversation, the solution maintains a chat history, which enables a conversational experience. This enables query disambiguation and follow-up questions. To reset a conversation and delete all chat history for this interaction, choose *Clear conversation *at the top of the chat window. Once the conversation has been cleared, a new session is created which re-enables editing of the settings.

Accessing and analyzing user collected feedback

As of v3.0.0, the Deployment Dashboard deploys a nested feedback stack which allows Text and Bedrock Agent usecases deployed with the Dashboard to have the functionality of feedback collection for the responses that the LLM/Agent generates. Particularly, users can provide a positive or negative feedback along with an optional comment. If the user provides a negative feedback, they can further select one of these negative categories: 'Inaccurate', 'Incomplete or insufficient', 'Harmful' and/or 'Other'.

Once the user provides the feedback, the feedback is stored in an S3 bucket partitioned by Use Case ID, year and month. The Use Case ID can be found in the Deployment Dashboard and the Feedback S3 bucket can be found in the outputs of the feedback nested stack of the Deployment Dashboard stack:

Depicts Deployment stack - Finding Feedback Bucket Name



The user feedback is sent as an API request containing a minimal set of information:

```
{
  "useCaseRecordKey": "a1b2c3d4-e5f6g7h8",
  "conversationId": "12345678-1234-1234-1234-123456789012",
  "messageId": "87654321-4321-4321-4321-210987654321",
  "rephrasedQuery": "What are the key features of the Generative AI Application Builder
 on AWS?",
  "sourceDocuments": [
    "s3://bucket-name/document1.pdf",
    "s3://bucket-name/document2.pdf"
  ],
  "feedback": "positive",
  "feedbackReason": [
    "Incomplete or insufficient"
  ],
  "comment": "The response was helpful but could include more details about important
 features."
}
```

This payload is then processed by a lambda using the `useCaseRecordKey` which identifies the correct configuration of a usecase at the time of deployment. This configuration is used to get specific details for the feedback such as the ConversationTable's name (contains all the conversations and human and AI message sequences) which is further used to retrieve the the actual `userInput` and `llmResponse`. Additional details are also attached to this feedback record such as the `agentId` and `agentAliasId` for a Bedrock Agent usecase, and `modelProvider`, `bedrockModelId`, etc. for a Text usecase using this configuration. For details on how to access this configuration, see Custom Feedback Mappings section below. Each incoming feedback request is stored as a JSON object and a sample feedback record can look like this for a Text usecase:


```
{
   "useCaseId": "12345678-1234-1234-1234-123456789012",
   "useCaseRecordKey": "c07a2e3b-2f31b1e0",
   "userId": "22345678-1234-1234-1234-123456789012",
   "conversationId": "dd51de5d-5af1-4ec6-91d2-aadf14352109",
   "messageId": "32345678-1234-1234-1234-123456789012",
   "userInput": "What are its key features?",
   "rephrasedQuery": "What are the key features of the Generative AI Application
 Builder on AWS?",
   "llmResponse": "Generative AI Application Builder on AWS can help you build
 production ready enterprise chatbots rapidly.",
   "feedback": "negative",
   "feedbackReason": [
      "Incomplete or insufficient"
   ],
   "comment": "The response was helpful but could include more details about important
 features.",
   "timestamp": "2025-05-22T18:48:08.340Z",
   "feedbackId": "42345678-1234-1234-1234-123456789012",
   "useCaseType": "Text",
   "modelProvider": "Bedrock",
   "bedrockModelId": "amazon.nova-lite-v1:0",
   "ragEnabled": "false"
}
```


or like this for a Bedrock Agent usecase:


```
{
   "useCaseId": "12345678-1234-1234-1234-123456789012",
   "useCaseRecordKey": "c07a2e3b-2f31b1e0",
   "userId": "22345678-1234-1234-1234-123456789012",
```

```
   "conversationId": "dd51de5d-5af1-4ec6-91d2-aadf14352109",
   "messageId": "32345678-1234-1234-1234-123456789012",
   "userInput": "What are its key features?",
   "llmResponse": "Generative AI Application Builder on AWS can help you build
 production ready enterprise chatbots rapidly.",
   "feedback": "negative",
   "feedbackReason": [
      "Incomplete or insufficient"
   ],
   "comment": "The response was helpful but could include more details about important
 features.",
   "timestamp": "2025-05-22T18:48:08.340Z",
   "feedbackId": "42345678-1234-1234-1234-123456789012",
   "useCaseType": "Agent",
   "agentId": "AHFXUJCAK1",
   "agentAliasId": "KSEDKOS0BL"
}
```

This feedback can then be used for further processing, analyzing and model re-training/feedback loops. You can also add custom mappings to enhance the feedback record being stored in the feedback lambda.

Custom Feedback Mappings

The Deployment Dashboard contains a `LLMConfigTable` which can be found in the stack outputs of the Deployment Dashboard stack with the key `LLMConfigTableName`. `LLMConfigTable` contains the configurations for each usecase based on the settings selected by the admin while deploying the usecase through the Deployment Dashboard wizard. Each usecase configuration is identified by its `useCaseRecordKey`. Here is a sample usecase configuration record in the `LLMConfigTable`:

```
{
   "key": "2dd76cfa-bc1a14da",
   "config": {
      "ConversationMemoryParams": {
        ...
      },
      "FeedbackParams": {
         "CustomMappings": {
            "NumberOfDocs": "$.KnowledgeBaseParams.NumberOfDocs",
            "ScoreThreshold": "$.KnowledgeBaseParams.ScoreThreshold"
         },
```

```
         "FeedbackEnabled": true
      },
      "IsInternalUser": "true",
      "KnowledgeBaseParams": {
         "KendraKnowledgeBaseParams": {
            "ExistingKendraIndexId": "d2831033-667f-4539-ab28-e6c7c7c5988b",
            "RoleBasedAccessControlEnabled": false
         },
         "KnowledgeBaseType": "Kendra",
         "NumberOfDocs": 5,
         "ReturnSourceDocs": false,
         "ScoreThreshold": 0.3
      },
      "LlmParams": {
         "BedrockLlmParams": {
            "BedrockInferenceType": "QUICK_START",
            "ModelId": "amazon.nova-lite-v1:0"
         },
         "ModelParams": {},
         "ModelProvider": "Bedrock",
         "PromptParams": {
            ...
         },
         "RAGEnabled": true,
         "Streaming": false,
         "Temperature": 0.1,
         "Verbose": false
      },
      "UseCaseName": "test-rag-usecase",
      "UseCaseType": "Text"
   }
}
```

If feedback is enabled for a usecase, this configuration will contain a `FeedbackParams` object which allows a `CustomMappings` object inside it that can specify the JSONPaths for all the additional fields to be added to the feedback JSON record stored in the feedback S3 bucket. For example, for the above sample usecase configuration, the CustomMappings contains `NumberOfDocs` and `ScoreThreshold` JSONPaths additionally in the `CustomMappings` object which start with `config` as the root of the JSONPath. With this configuration, each JSON record stored in the feedback S3 bucket will start getting these 2 additional values apart from the fields that have already been provided.

Analyzing feedback data

The feedback data is stored in S3 as JSON objects. Here are some approaches to make this feedback data more accessible and actionable:

Using AWS Glue and Amazon Athena

AWS Glue and Amazon Athena provide a serverless way to catalog, query, and analyze your feedback data.

AWS Glue allows you to create an AWS Glue crawler that inspects the data in an S3 bucket, infers its schema, and records all the relevant metadata in a catalog. Post that, services like Amazon Athena can be used to query the data.

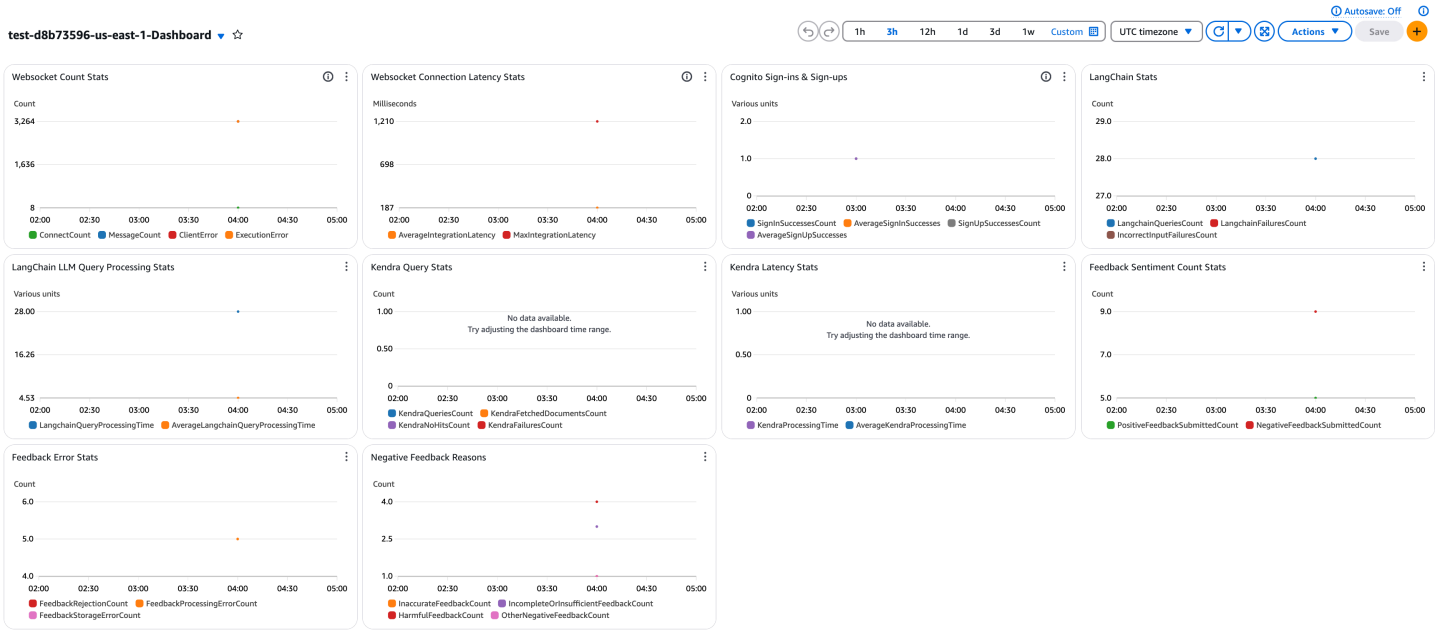
You can refer AWS Athena Documentation on steps for connecting the feedback S3 bucket with Amazon Athena using AWS Glue Data Catalog. You can also use some of Glue's more powerful features to perform Extract Transform & Load (ETL) jobs on this data and transform it into a format that suits your analytics or model re-training use cases. With Glue, you can perform operations such as filtering the records with certain feedback types, filling out any missing information, and you can also load this data into another storage location such as another S3 bucket or a different AWS data store.

> **Note**
>
> Depending upon your use case, consider scheduling the Glue crawler to run periodically (e.g., weekly) rather than nightly to optimize costs as feedback data can be sparse.

Using the solution's CloudWatch Dashboards

You also have access to a **CloudWatch Dashboard** packaged with the solution that can provide you trends for positive and negative feedback, negative feedback reason categories, etc. on a per use-case basis. You can find this dashboard using your usecase name in *Dashboards* inside the AWS CloudWatch console:

Depicts Usecase CloudWatch Dashboard



You can also build additional widgets in this Dashboard or create Amazon Quick Sight dashboards.

Best practices for feedback data analysis

- **Implement data lifecycle policies** on your S3 bucket to archive older feedback data to lower-cost storage tiers
- **Create separate analysis for each use case** to identify model-specific improvement opportunities
- **Establish feedback thresholds** that trigger alerts when negative feedback exceeds acceptable levels
- **Export critical insights** periodically for sharing with stakeholders and model improvement teams

Viewing operation metrics for a deployment

The Deployment dashboard and use case stacks each come with their own CloudWatch dashboard tracking various operational metrics of the solution. You can use these CloudWatch dashboards to help compare different deployments. To access the dashboards:

1. Navigate to the CloudWatch console.
2. Search for the pre-built dashboards either by looking up the stack name, or universally unique identifier (UUID).

For example, the Text use case comes with graphs tracking the number of WebSocket connections, the number of user sign ins and sign ups, the amount of time the LLM took to process a completion, and so on. Customers can use these graphs to compare various _quantitative _metrics of a deployment.

Example

It is difficult to compare the *qualitative* results of various models applied to different use cases. Use the Clone feature to spin up multiple deployments quickly so that you can compare the outputs side by side.

Access CloudWatch Logs insights

This solution logs error, warning, informational, and debugging messages for the Lambda functions. To choose the type of messages to log:

1. Locate the applicable function in the AWS Lambda console.
2. Add a **POWERTOOLS_LOG_LEVEL** environment variable.
3. Set the variable to the applicable type of message.

For further instructions, see Create Lambda environment variables in the *AWS Lambda Developer Guide*.

The following table lists the types of log levels that you can choose from.

Level	Description
ERROR	Logs include information on anything that causes an operation to fail.
WARNING	Logs include information on anything that could potentially cause inconsistencies in the function but might not necessarily cause the operation to fail. Logs also include ERROR messages.

Level	Description
INFO	Logs include high-level information about how the function is operating. Logs also include ERROR and WARNING messages.
DEBUG	Logs include information that might be helpful when debugging a problem with the function. Logs also include ERROR, WARNING, and INFO messages.

Use the following procedure to add CloudWatch Logs insights to this solution.

1. Identify the relevant log groups:
   a. Sign in to the AWS CloudFormation console.
   b. Choose your target stack.
   c. Select the **Resources** tab and search for your target Lambda functions.
   d. Sign in to the AWS Lambda console and choose each of your target Lambda functions.
   e. For each of your target Lambda functions, select the **Monitor** tab and choose **View CloudWatch Logs**.
   f. Copy the names of the log groups that you want to extract insights from.
2. Navigate to the Amazon CloudWatch console.
3. On the navigation menu, under **Logs**, choose **Logs Insights**.
4. On the **Logs Insights** page, choose the **Logs** tab.
5. Search for log group names from step 1.
6. Copy one of the following example queries and paste it into the query field:
   a. To identify all client exceptions:

```
fields @message
|filter @message like /(?i)Exception/|stats count(*) as exceptionCount by @message
```

   b. To retrieve count of invocations by function name:

```
stats count(*) by function_name
```

c. To retrieve count of invocations over five-minute intervals:

```
stats count(*) as invocations by bin(5m)
```

d. To retrieve all AWS X-Ray trace IDs:

```
filter @message like "XRAY TraceId"
|parse @message "XRAY TraceId: * " as traceId|stats count(*) by traceId
```

e. To retrieve logs relating to a specific X-Ray Trace ID:

```
filter @message like "your-traceid-here"
```

f. To retrieve unauthorized WebSocket errors:

```
fields
@ingestionTime,
@log,
@logStream,
@message,
@requestId,
@timestamp,
errorMessage,
errorType
|filter @message like /Unauthorized/ and @message like /websocket/|sort @timestamp
 desc
```

g. To retrieve count of metrics published:

```
filter @message like "CloudWatchMetrics"
|parse @message /"Metrics":\s*\[(?<metrics>.*?)\]/|stats count(*) as metric_count
 by metrics
```

Developer guide

This section provides the source code for the solution, an integration guide, a customization guide, and API reference.

Source code

Visit our GitHub repository to download the source files for this solution and to share your customizations with others.

The Generative AI Application Builder on AWS templates are generated using the AWS Cloud Development Kit (AWS CDK). See the README.md file for additional information.

Integration guide

The entire solution is designed to be easily extensible. The orchestration layer of this solution is built using LangChain. You can add any model provider, knowledge base, or conversation memory type supported by LangChain (or a third party that provides LangChain connectors for these components) to this solution.

Expanding supported LLMs

To add another model provider, such as a custom LLM provider, you must update the following three components of the solution:

1. Create a new `TextUseCase` CDK stack, which deploys the chat application configured with your custom LLM provider:
   a. Clone this solution's GitHub repository, and set up your build environment by following the instructions provided in the README.md file.
   b. Copy (or create new) the `source/infrastructure/lib/bedrock-chat-stack.ts` file, paste it to the same directory, and rename it to `custom-chat-stack.ts`.
   c. Rename the class in the file to a suitable one, such as `CustomLLMChat`.
   d. You can choose to add a Secrets Manager secret to this stack, which stores your credentials for your custom LLM. You can retrieve these credentials during model invocation in the chat Lambda layer discussed in the next paragraph.

2. Build and attach a Lambda layer containing the Python library of the model provider to be added. For an Amazon Bedrock use case chat application, the `langchain-aws` Python library contains the custom connectors on top of the LangChain package to connect to the AWS model providers (Amazon Bedrock and SageMaker AI), knowledge bases (Amazon Kendra and Amazon Bedrock Knowledge Bases), and memory types (such as DynamoDB). Similarly, other model providers have their own connectors. This layer helps you attach this model provider's Python library so that you can use these connectors in the chat Lambda layer, which invokes the LLM (step 3). In this solution, a custom asset bundler is used to build Lambda layers, which are attached using CDK aspects. To create a new layer for the custom model provider library:
   a. Navigate to the `LambdaAspects` class in the `source/infrastructure/lib/utils/lambda-aspects.ts` file.
   b. Follow the instructions on how to extend the functionality of the Lambda aspects class provided in the file (such as adding the `getOrCreateLangchainLayer` method). To use this new method (for example, `getOrCreateCustomLLMLayer`), also update the `LLM_LIBRARY_LAYER_TYPES` enum in the `source/infrastructure/lib/utils/constants.ts` file.
3. Extend the `chat` Lambda function to implement a builder, client, and handler for the new provider.

   The `source/lambda/chat` contains the LangChain connections for different LLMs along with the supporting classes to build these LLMs. These supporting classes follow Builder and Object Oriented design patterns to create the LLM.

   Each handler (for example, `bedrock_handler.py`) first creates a *client*, checks the environment for required environment variables, and then calls a `get_model` method to get the LangChain LLM class. The generate method is then called to invoke the LLM and get its response. LangChain currently supports streaming functionality for Amazon Bedrock, but not SageMaker AI. Based on streaming or non-streaming functionality, appropriate WebSocket handler (`WebsocketStreamingCallbackHandler` or `WebsocketHandler`) is called to send the response back to the WebSocket connection using the `post_to_connection` method.

   The `clients/builder` folder contains the classes which help build an LLM Builder using Builder pattern. First, a `use_case_config` is retrieved from a DynamoDB configurations store, which stores the details on what type of knowledge base, conversation memory, and model to construct. It also contains relevant model details such as model parameters and prompts. The Builder then helps in following the steps for creating a knowledge base, creating a conversation memory to maintain conversation context for LLM, setting the appropriate LangChain callbacks

for streaming and non-streaming cases, and creating an LLM model based on the provided model configurations. The DynamoDB configuration is stored at the time of use case creation when you deploy a use case from the Deployment dashboard (or when it is provided by the users in standalone use case stack deployments without the Deployment dashboard).

The `clients/factories` subfolder helps set the appropriate conversation memory and knowledge base class, based on the LLM configruation. This enables easy extension to any other knowledge base or memory types that you want your implementation to support.

The `shared` subfolder contains specific implementations of knowledge base and conversation memory which are instantiated inside the factories by the builder. It also contains Amazon Kendra and Amazon Bedrock Knowledge Base retrievers called within LangChain to retrieve documents for the RAG use cases, along with callbacks, which are used by the LangChain LLM model.

The LangChain implementations use LangChain Expression Language (LCEL) to compose conversation chains together. `RunnableWithMessageHistory` class is used to maintain conversation history with custom LCEL chains, enabling functionalities such as returning source documents and using the rephrased (or disambiguated) question sent to the knowledge base to also be sent to the LLM.

To create your own implementation of a custom provider, you can:

a. Copy the `bedrock_handler.py` file and create your custom handler (for example, `custom_handler.py`), which creates your custom client (for example, `CustomProviderClient`) (specified in the following step.)

b. Copy `bedrock_client.py` in the clients folder. Rename it to `custom_provider_client.py` (or your specific model provider name, such as `CustomProvider`). Name the class within it appropriately, such as `CustomProviderClient` which inherits `LLMChatClient`.

   You can use the methods provided by `LLMChatClient` or write your own implementations to override these.

   The `get_model` method builds a `CustomProviderBuilder` (see the following step), and calls the `construct_chat_model` method that constructs the chat model using builder steps. This method acts as the *Director* in the builder pattern.

c. Copy `clients/builders/bedrock_builder.py` and rename it to `custom_provider_builder.py` and the class within it to `CustomProviderBuilder` that

inherits LLMBuilder (`llm_builder.py`). You can use the methods provided by LLMBuilder or write your own implementations to override these. The builder steps are called in sequence inside the client's `construct_chat_model` method, such as `set_model_defaults`, `set_knowledge_base`, and `set_conversation_memory`.

The `set_llm_model` method would create the actual LLM model using all of the values that are set using the methods called before it. Specifically, you can create a RAG (`CustomProviderRetrievalLLM`) or non-RAG (`CustomProviderLLM`) LLM, based on the `rag_enabled variable` that is retrieved from the LLM configuration in DynamoDB.

This configuration is fetched in the `retrieve_use_case_config` method in the `LLMChatClient` class.

d. Implement your `CustomProviderLLM` or `CustomProviderRetrievalLLM` implementation in the `llm_models` subfolder based on whether you require RAG or non-RAG use case. Most functionalities to implement these models are provided in their `BaseLangChainModel` and `RetrievalLLM` classes respectively, for non-RAG and RAG use cases.

You can copy the `llm_models/bedrock.py` file and make the necessary changes to call the LangChain model that refer to your custom provider. For example, Amazon Bedrock uses a `ChatBedrock` class to create a chat model using LangChain.

The generate method generates the LLM response using the LangChain LCEL *chains*.

You can also use the `get_clean_model_params` method to sanitize the model parameters per LangChain or your model requirements.

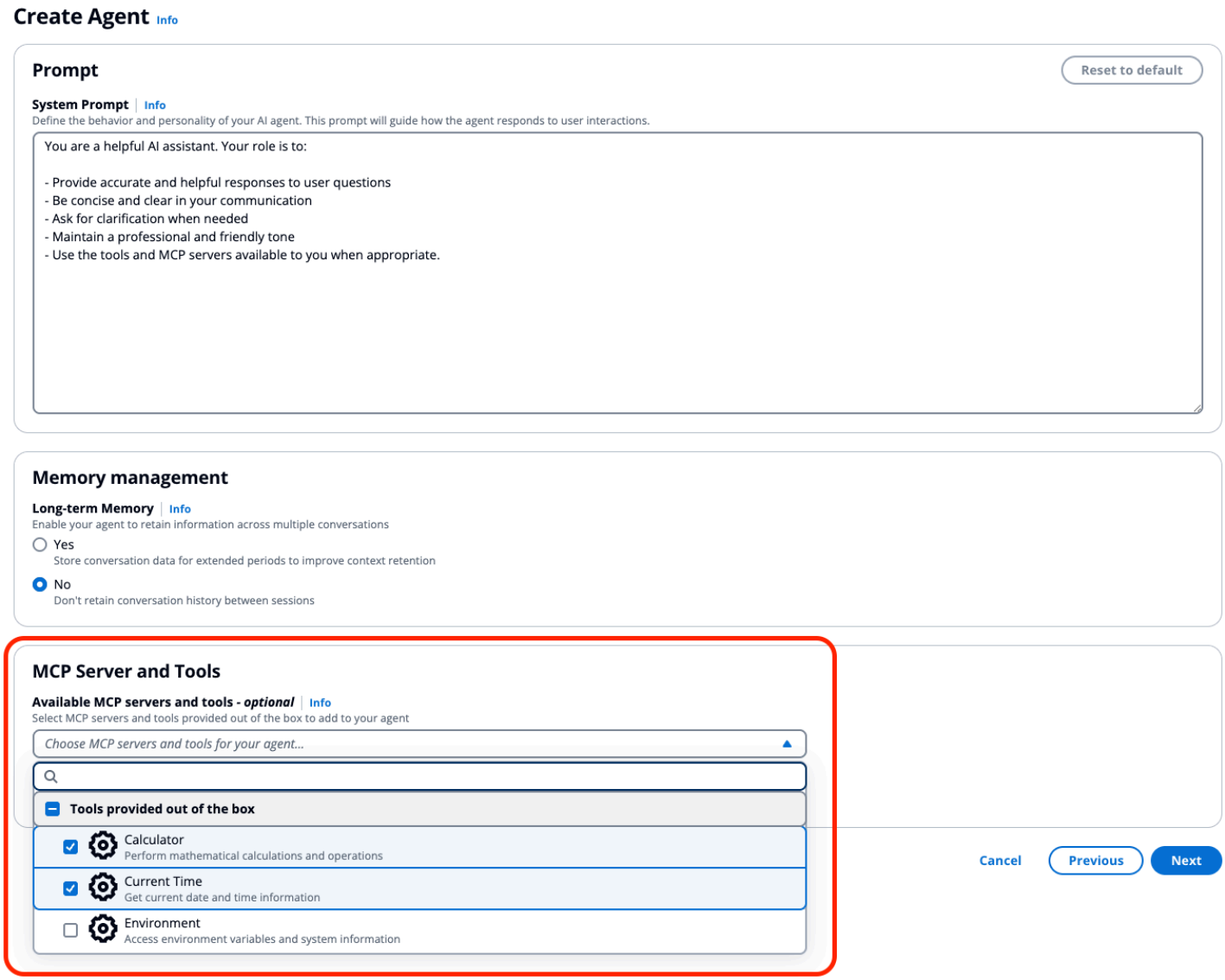
Expanding supported Strands tools

The Solution enables you to build and deploy MCP servers, AI agents, and multi-agent workflows. Within the Agent Builder experience, you can attach MCP servers to give your agents additional capabilities. In addition to MCP servers, you can leverage built-in tools provided by Strands (the underlying framework used by the solution).

Out of the box, the solution comes pre-configured with the following Strands tools:

- Current Time (enabled by default)
- Calculator (enabled by default)
- Environment

MCP Server and Tools selection in the Agent Builder wizard showing built-in Strands tools



To extend your agents with additional Strands tools, follow the four-step process outlined in this section.

Step 1: Find the Strands tool

Browse the available Strands tools to identify the tool you want to use. Each tool has specific capabilities and configuration requirements.

For example, to add Amazon Bedrock Knowledge Base retrieval capabilities, you would use the retrieve tool.

Step 2: Update the SSM parameter

To make a tool available in the Agent Builder deployment UI, update the AWS Systems Manager Parameter Store parameter that defines which Strands tools are supported.

1. Navigate to the AWS Systems Manager Parameter Store in your AWS account.
2. Locate the parameter: `/gaab/<stack-name>/strands-tools`
3. Add your tool configuration to the end of the existing list using the following JSON structure:

```
{
  "name": "Bedrock KB Retrieve",
  "description": "Retrieve information from Bedrock Knowledge Base",
  "value": "retrieve",
  "category": "AI",
  "isDefault": false
}
```

Field	Description
name	Display name shown in the Agent Builder UI
description	Brief description of the tool's functionality
value	The exact tool name as defined in the Strands tools package
category	Organizational category for grouping tools in the UI
isDefault	Whether the tool should be enabled by default for new agents

Step 3: Configure environment variables

Many Strands tools require environment variables for configuration. You can set these variables in two ways:

Option 1: Direct configuration on AgentCore Runtime

Update the deployed agent directly on Amazon Bedrock AgentCore Runtime with the required environment variables.

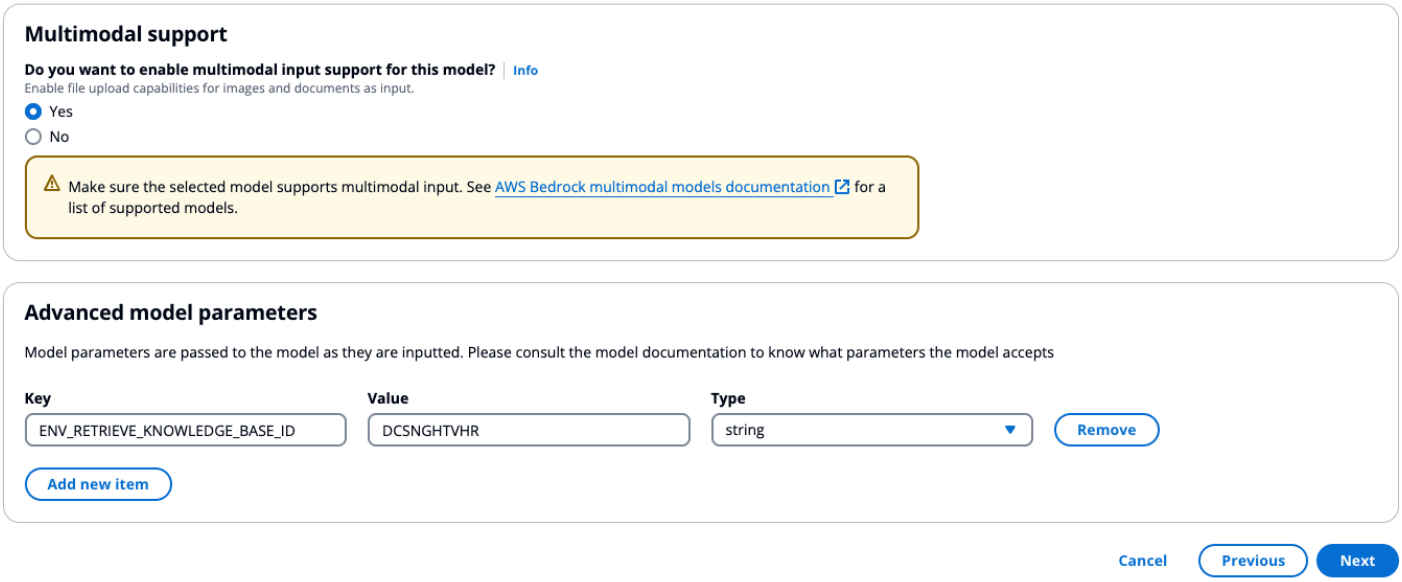
Option 2: Model Parameters in the deployment wizard

Add environment variables during the Model selection step in the Agent Builder wizard using the Model Parameters section. Environment variables that follow the naming convention `ENV_<ALL_CAPS_TOOL_NAME>_<env_variable_name>` will automatically be loaded at runtime into the agent's execution environment as `<env_variable_name>`.

For example:

- `ENV_RETRIEVE_KNOWLEDGE_BASE_ID` becomes `KNOWLEDGE_BASE_ID`
- `ENV_RETRIEVE_MIN_SCORE` becomes `MIN_SCORE`

Advanced model parameters section showing ENV_RETRIEVE_KNOWLEDGE_BASE_ID configuration



Refer to the specific tool's documentation or source code to identify required environment variables. For the retrieve tool, you can find configuration options in the source code.

Step 4: Add IAM permissions

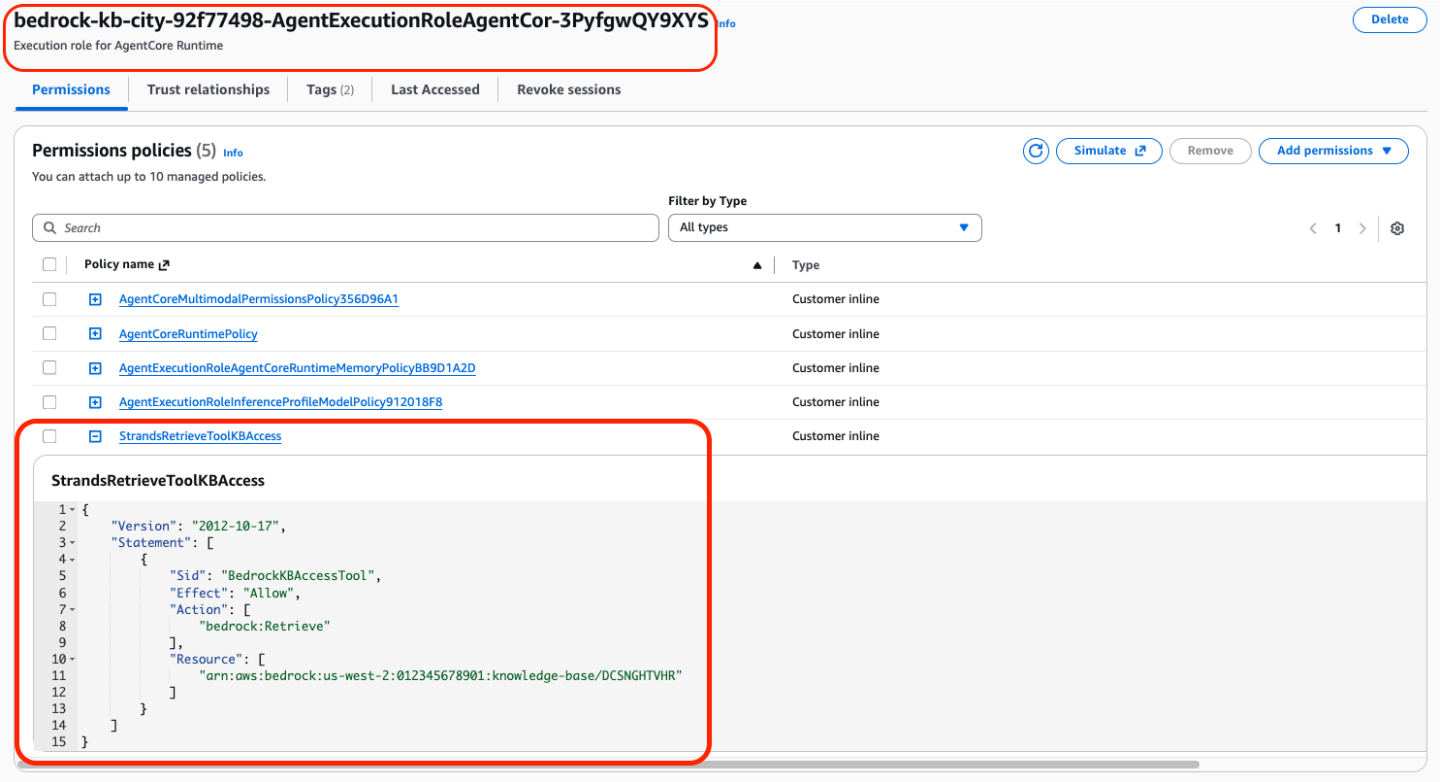
Manually add any necessary IAM permissions to your AgentCore Runtime execution role to allow the agent to use the tool.

For example, to use the retrieve tool with Amazon Bedrock Knowledge Bases:

1. Navigate to the IAM console in your AWS account.
2. Locate the AgentCore Runtime execution role for your agent.
3. Add the following permission:

```
{
  "Effect": "Allow",
  "Action": "bedrock:Retrieve",
  "Resource": "arn:aws:bedrock:region:account-id:knowledge-base/knowledge-base-id"
}
```

IAM console showing the StrandsRetrieveToolKBAccess policy attached to the AgentCore Runtime execution role

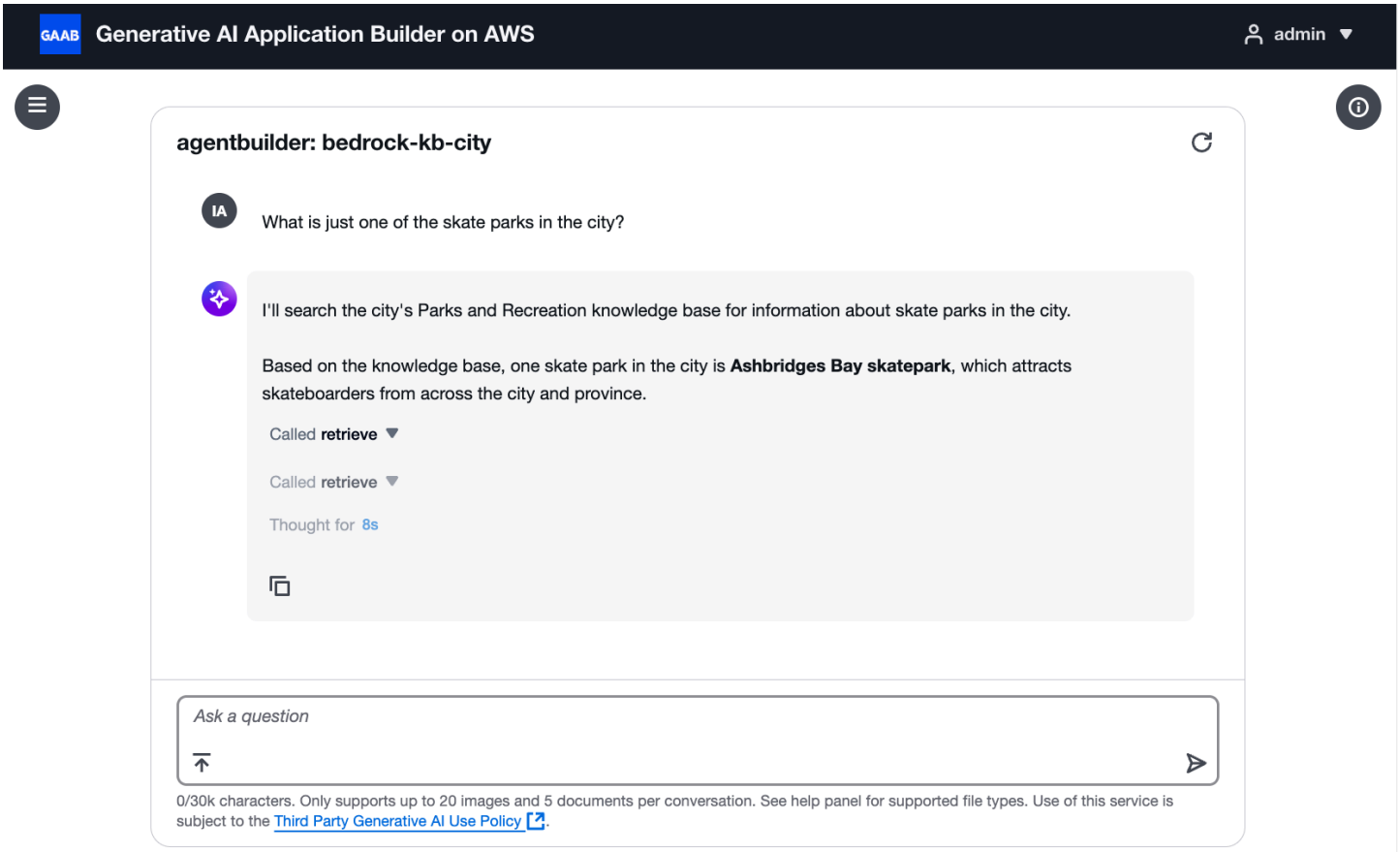


The specific permissions required will vary based on the tool. Consult the tool's documentation and AWS service documentation to determine the appropriate IAM permissions.

Step 5: Test the agent

After completing the configuration steps, test your agent to verify the tool is working correctly. You should see tool invocations in the agent's execution logs and responses.

Agent successfully using the retrieve tool to answer a question about skate parks



> **Note**
>
> For a complete list of available Strands tools and their capabilities, refer to the Strands Community Tools documentation.

Expanding supported knowledge bases and conversation memory types

To add your implementations of conversation memory or knowledge base, add the required implementations in the `shared` folder and then edit the factories and appropriate enumerations to create an instance of these classes.

When you supply the LLM configuration, which is stored inside the parameter store, the appropriate conversation memory and knowledge base will be created for your LLM. For example, when the `ConversationMemoryType` is specified as DynamoDB, an instance of `DynamoDBChatMessageHistory` (available inside `shared_components/memory/ddb_enhanced_message_history.py`) is created. When the `KnowledgeBaseType` is specified as Amazon Kendra, an instance of `KendraKnowledgeBase` (available inside `shared_components/knowledge/kendra_knowledge_base.py`) is created.

Building and deploying the code changes

Build the program with the `npm run build` command. Once any errors are resolved, run `cdk synth` to generate the template files and all the Lambda assets.

1. You can use the `0/stage-assets.sh` script to manually stage any generated assets to the staging bucket in your account.
2. Use the following command to deploy or update platform:

```
cdk deploy DeploymentPlatformStack --parameters AdminUserEmail='admin-
email@amazon.com'
```

   Any additional AWS CloudFormation parameters should also be supplied along with the **AdminUserEmail** parameter.

Customization guide

Managing Cognito user pool

When the Deployment dashboard is deployed, an Amazon Cognito user pool along with an admin user are created to provide authentication for the application. This user pool is shared across the Deployment dashboard and all use cases. The admin user created on deployment of the dashboard is automatically granted access to all use cases deployed using the dashboard. This mechanism is provided via Amazon Cognito user pool groups.

When a use case is deployed from the dashboard, if an email is provided, a user will be created in the shared user pool, along with a user group named for the specific use case. The newly created user is then added to the group, granting the user access to the use case.

If you wish to add an additional user to a given use case, this can be achieved by creating a user in the Cognito user pool and adding them to the group(s) corresponding to the use case(s) you want the user to have access to. For a step-by-step guide, see Creating a new user in the AWS Management Console.

Similarly, if you want to create additional admin users, you must create a new user and add them to the **Admin** group in the user pool.

The user names are created by taking the portion of the provided email before the `@`, and appending the generated use case UUID (or `-admin` in the case of the admin user).

In the **Groups** tab, you can see that an **Admin** group and a group for each use case have been automatically created using the name of the use case (as provided in the wizard) and the use case UUID.

API reference

This section provides API references for the solution.

Deployment dashboard

REST API	HTTP method	Functionality	Authorized callers
`/deployments`	`GET`	Get all deployments.	Amazon Cognito authenticated JWT token
`/deployments`	`POST`	Creates a new use case deployment.	Amazon Cognito authenticated JWT token
`/deployments/{useCaseId}`	`GET`	Gets deployment details for a single deployment.	Amazon Cognito authenticated JWT token
`/deployments/{useCaseId}`	`PATCH`	Updates a given deployment.	Amazon Cognito authenticated JWT token

REST API	HTTP method	Functionality	Authorized callers
`/deployments/{useCaseId}`	`DELETE`	Deletes a given deployment.	Amazon Cognito authenticated JWT token
`/model-info/use-case-types`	`GET`	Gets the available use case types for the deployment	Amazon Cognito authenticated JWT token
`/model-info/{useCaseType}/providers`	`GET`	Gets the available model providers for the given use case type	Amazon Cognito authenticated JWT token
`/model-info/{useCaseType}/{providerName}`	`GET`	Gets the IDs of the models available for a given provider and use case type	Amazon Cognito authenticated JWT token
`/model-info/{useCaseType}/{providerName}/{modelId}`	`GET`	Gets the info about the given model, including default parameters.	Amazon Cognito authenticated JWT token

Note

OpenAPI and Swagger files can also be exported from API Gateway for easier integration with the API. See Export a REST API from API Gateway.

POST and PATCH Payloads

See below for an example of a POST payload to the `/deployments` endpoint, which will create a new use case.

```
{
 "UseCaseName": "usecase1",
```

```
"UseCaseDescription": "Description of the use case to be deployed. For display
purposes", // optional
"DefaultUserEmail": "placeholder@example.com", // optional, if not provided, the
Cognito Group and User will not be created
"DeployUI": true, // optional
"VpcParams": {
"VpcEnabled": true,
"CreateNewVpc": false,
// provide these if not creating new vpc
"ExistingVpcId": "vpc-id",
"ExistingPrivateSubnetIds": ["subnet-1", "subnet-2"],
"ExistingSecurityGroupIds": ["sg-1", "sg-2"]
},
"ConversationMemoryParams": {
"ConversationMemoryType": "DynamoDB",
"HumanPrefix": "user", // optional
"AiPrefix": "ai", // optional
"ChatHistoryLength": 10 // optional
},
"KnowledgeBaseParams": {
"KnowledgeBaseType": "Bedrock",
// one of the following based on selected provider
"BedrockKnowledgeBaseParams": {
"BedrockKnowledgeBaseId": "my-bedrock-kb",
"RetrievalFilter": {}, // optional
"OverrideSearchType": "HYBRID" // optional
},
"KendraKnowledgeBaseParams": {
"AttributeFilter": {}, // optional
"RoleBasedAccessControlEnabled": true, // optional
"ExistingKendraIndexId": "12345678-abcd-1234-abcd-1234567890ab",
// provide the following in place of ExistingKendraIndexId if you want the solution to
deploy an index for you
"KendraIndexName": "index",
"QueryCapacityUnits": 1, // optional
"StorageCapacityUnits": 1, // optional
"KendraIndexEdition": "DEVELOPER" // optional
},
"NoDocsFoundResponse": "Sorry, I couldn't find any relevant information for your
query.", // optional
"NumberOfDocs": 3, // optional
"ScoreThreshold": 0.7, // optional
"ReturnSourceDocs": true // optional
},
```

```
 "LlmParams": {
 "ModelProvider": "Bedrock | SAGEMAKER",
 // one of the following based on selected provider
 "BedrockLlmParams": {
 "ModelId": "model-id", // use this for on demand models. Can't use with ModelArn
 "ModelArn": "model-arn", // use this for provisioned/custom models. Can't use with
 ModelId,
 "InferenceProfileId": "profile-id"
 "GuardrailIdentifier": "arn:aws:bedrock:us-east-1:123456789012:guardrail/my-
guardrail", // optional
 "GuardrailVersion": "1" // optional. Required if GuardrailIdentifier provided.
 },
 "SageMakerLlmParams": {
 "EndpointName": "some-endpoint",
 "ModelInputPayloadSchema": {},
 "ModelOutputJSONPath": "$."
 },
 // optional. Passes on arbitrary params to the underlying LLM.
 "ModelParams": {
 "param1": {
 "Value": "value1",
 "Type": "string"
 },
 "param2": {
 "Value": 1,
 "Type": "integer"
 }
 },
 // optional
 "PromptParams": {
 "PromptTemplate": "some template",
 "UserPromptEditingEnabled": true,
 "MaxPromptTemplateLength": 1000,
 "MaxInputTextLength": 1000,
 "DisambiguationPromptTemplate": "some disambiguation template",
 "DisambiguationEnabled": true
 },
 "Temperature": 1.0, // optional
 "Streaming": true, // optional
 "RAGEnabled": true, // optional. Must be true if providing KnowledgeBaseParams above.
 "Verbose": false // optional
 },
 "AgentParams": {
 "AgentType": "Bedrock",
```

```
 "BedrockAgentParams": {
 "AgentId": "agent-id",
 "AgentAliasId": "alias-id",
 "EnableTrace": true
 }
 },
 // optional
 "AuthenticationParams": {
 "AuthenticationProvider": "Cognito",
 "CognitoParams": {
 "ExistingUserPoolId": "user-pool-id",
 "ExistingUserPoolClientId": "client-id" // optional. If not provided, the solution
 will create a client for you in the provided pool
 }
 }

}
```

For updates, the structure is the same as above with some caveats:

- The use case name cannot be changed
- A use case can only change security groups and subnets once it has been deployed in a VPC. The VPC itself cannot be changed.
- If a Kendra index was created for you as a knowledge base, you cannot change the configuration of that index (for example, KendraIndexName, QueryCapacityUnits)

Shared Use Case APIs

The following REST API endpoints are available for both Text and Bedrock Agent use cases:

REST API	HTTP method	Functionality	Authorized callers
`/details/{useCaseConfigKey}`	`GET`	Gets configuration details for a specific use case.	Amazon Cognito authenticated JWT token

WebSocket API	Functionality	Authorized callers
/$connect	Initiate WebSocket connection and authenticate user.	Amazon Cognito authenticated JWT token
/$disconnect	Endpoint called when a WebSocket connection has been disconnected.	Amazon Cognito authenticated JWT token

Use Case Details API

The details API endpoint retrieves information about a specific use case:

```
GET /details/{useCaseConfigKey}
```

This endpoint returns the configuration details for a specific use case, including model parameters, knowledge base settings, and other deployment information. It requires an Amazon Cognito authenticated JWT token for authorization.

Text use case

WebSocket API	Functionality	Authorized callers
/sendMessage	Sends user's chat message to the WebSocket for processing with the configured LLM experience.	Amazon Cognito authenticated JWT token

REST API	HTTP method	Functionality	Authorized callers
/feedback/{useCaseId}	POST	Submits user feedback for a specific use case.	Amazon Cognito authenticated JWT token

sendMessage Payloads

If you're directly integrating with the `/sendMessage` API, you must adhere to the following request and response payload formats.

Request Payload

```
{
 "action": "sendMessage",
 "question": "the message to send to the api",
 "conversationId": "", // If not provided, a new conversation will be created, with the
 conversationId returned in the response. All subsequent messages in that conversation
 (where history is retained), should provide the conversationId there.
 "promptTemplate": "", // Optional. Overrides the configured prompt
 "authToken": "XXXX" // Optional. accessToken from cognito flow. Required for RAG with
 RBAC
}
```

Parameter Name	Type	Description
action	`String`	Currently we only support the "sendMessage" action on the WebSocket
question	`String`	The user input to send to the LLM
conversationId	`String`	A UUID identifying the conversation. If not provided, a new conversation will be created, with the conversat ionId returned in the response. All subsequent messages in that conversation (where you wish for history/c ontext to be retained), should provide the conversationId there.
promptTemplate	`String` [Optional]	Overrides the prompt template for this message. If

Parameter Name	Type	Description
		empty or not provided, will default to the prompt set at deployment time. Must have the proper placeholders specified for the given configuration (i.e. {history} and {input} for non-RAG Sagemaker AI deployments, with the addition of {context} if using RAG for all deployments.
authToken	`String` [Optional]	accessToken as obtained from the cognito auth flow. This is required when invoking a chat websocket endpoint configured for RAG with Role Based Access Control (RBAC). The cognito:groups claim list in this JWT token is used to control access to documents in the Kendra index. This parameter is **not** required for non-RAG use cases. It is also **not** required for RAG use cases that has RBAC disabled.

Response Payloads

Question Response

The WebSocket API will respond with 1 (if streaming is disabled) or many (if streaming is enabled) JSON objects structured as follows for each query.

```
{
 "data": "some data",
```

```
 "conversationId": "id",
}
```

Parameter Name	Type	Description
data	`String`	A chunk of the response from the LLM if streaming is enabled, or the entire response. If using streaming , a response of this format with the data content being *END_CONVERSATION* will be sent to indicate the end of the response to a single question.
conversationId	`String`	The ID of the conversat ion this sourceDocument response belongs to.

Source Document Response

If you have configured your RAG use case to return source documents, you will also receive the following payload at the end of every response for each source document used to create the response.

```
{
 "sourceDocument": {
 "excerpt": "some excerpt from the",
 "location": "s3://fake-bucket/test.txt",
 "score": 0.500,
 "document_title": null,
 "document_id": null,
 "additional_attributes": null
 },
 "conversationId": "some-id"
}
```

Parameter Name	Type	Description
excerpt	`String`	An excerpt from the source document.
location	`String`	Location of the source document. This will depend on the data sources used and type of knowledge base, but could be things like s3 URIs or websites.
score	`Number` | `String`	The confidence that the document corresponds to the question asked. This will be a float from 0 to 1 for Bedrock, and a string (e.g. HIGH, LOW, etc.) for Kendra.
document_title	`String`	Title of the returned source document. Only available when using Kendra.
document_id	`String`	ID of the returned source document. Only available when using Kendra.
additional_attributes	`String`	This field will contain all additional attributes on the document as customized on your knowledge base at ingestion.
conversationId	`String`	The ID of the conversat ion this sourceDocument response belongs to.

Feedback API Payload

Below is an example of a POST payload to the `/feedback/{useCaseId}` endpoint, which will submit user feedback for a specific use case:

```
{
  "useCaseRecordKey": "12345678-12345678",
  "conversationId": "12345678-1234-1234-1234-123456789012",
  "messageId": "12345678-1234-1234-1234-123456789012",
  "feedback": "positive",
  "feedbackReason": ["accurate", "helpful"],
  "comment": "This response was very helpful.",
  "rephrasedQuery": "What are the key features of Amazon Bedrock?",
  "sourceDocuments": [
    "s3://bucket-name/document1.pdf",
    "s3://bucket-name/document2.pdf"
  ]
}
```

Bedrock Agent use case

WebSocket API	Functionality	Authorized callers
`/invokeAgent`	Sends user's message to the WebSocket for processing with the configured agent.	Amazon Cognito authentic ated JWT token

invokeAgent Payloads

If you're directly integrating with the `/invokeAgent API`, you must adhere to the following request and response payload formats.

Request payload

```
{
"action": "invokeAgent",
"inputText": "User query to the agent",
"conversationId": "", // Optional. Empty conversationId implies a new conversation.
 When not provided, a new conversationId will be created and returned with the
```

```
 response. All subsequent messages in the same conversation should provide the same
 conversationId (i.e. chat memory/history is maintained).
"authToken": "XXXX" // Optional. accessToken from cognito flow. If provided, it needs
 to be a valid JWT token associated with the user
}
```

Parameter name	Type	Description
action	`String`	We only support the `invokeAgent` action on the WebSocket.
inputText	`String`	The user input to send to the LLM.
conversationId	`String[Optional]`	A UUID that uniquely identifies the conversation. If you don't provide this value, the solution creates a new conversation and returns the **conversationId** in the response. All subsequent messages in that conversation (where you want to retain history and context) provide the **conversationId** there.
authToken	`String[Optional]`	**accessToken** as obtained from the Amazon Cognito auth flow. This parameter is **not** required. If you provide it, the JWT token will be validated. This helps make it easier for this solution to be extended.

Response payloads

Question response

The WebSocket API will respond with one (if streaming is disabled) or many (if streaming is enabled) JSON objects structured as follows for each query.

```
{
 "data" "some data",
 "conversationId": "id",
 }
```

Parameter name	Type	Description
data	`String`	The response from the agent invocation.
conversationId	`String`	The ID of the conversation.

Reference

This section includes information about data collection for this solution, pointers to related resources, and a list of builders who contributed to this solution.

Supported LLM providers

The solution can integrate with the following LLM providers:

1. Amazon Bedrock
   - Documentation: https://aws.amazon.com/bedrock/
   - Supported models:
     - Amazon
       - Nova Lite
       - Nova Micro
       - Nova Pro
     - AI21 Labs
       - Jamba 1.5 Mini
       - Jamba 1.5 Large
     - Anthropic
       - Claude v3 Haiku
       - Claude v3.5 Sonnet
       - Claude v3.7 Sonnet (through the use of inference profiles)
     - Cohere
       - Command R
       - Command R+
     - Deepseek
       - Deepseek-R1 (through the use of inference profiles)
     - Meta
       - Llama 3
       - Llama 3.2 (through the use of inference profiles)
     - Mistral AI

    - Mistral 7B Instruct
    - Mistral 8x7B Instruct
  - Cross-region inference
    - Ability to use inference profiles defined in the same Region as the Deployment dashboard

2. Amazon SageMaker AI
   - Documentation: https://aws.amazon.com/sagemaker/
   - Supported models: Text to Text models

For the latest model parameters, best practices, and recommended uses, refer to the documentation from the model providers.

Data collection

This solution sends operational metrics to AWS (the "Data") about the use of this solution. We use this Data to better understand how customers use this solution and related services and products. AWS's collection of this Data is subject to the AWS Privacy Notice.

Contributors

- Tarek Abdunabi
- Majd Arbash
- George Bearden
- Mukit Bin Momin
- Michael Connor
- Johny Duval
- Nihit Kasabwala
- Ahern Knox
- Simon Krol
- Michael Lin
- Tim Mekari
- Ibrahim Mohamed
- Omar Radwan Mohsen

- James Nixon
- Dekshitha Ravikumar
- Jae Shim
- Ajay Swamy
- Mohammed Taha
- Reet Takkar
- Dimitri Tchikatilov
- Jason Wreath
- Kamyar Ziabari

Revisions

Publication date: *October 2023 (last update: January 2025)*

Check the CHANGELOG.md file in the GitHub repository to see all notable changes and updates to the software. The changelog provides a clear record of improvements and fixes for each version.

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents AWS current product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. AWS responsibilities and liabilities to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Generative AI Application Builder on AWS is licensed under the terms of the Apache License Version 2.0.

Important

Generative AI Application Builder on AWS allows you to build and deploy generative artificial intelligence applications on AWS by engaging the generative AI model of your choice, including third-party generative AI models that you can choose to use that AWS does not own or otherwise have any control over ("Third-Party Generative AI Models"). Your use of the Third-Party Generative AI Models is governed by the terms provided to you by the Third-Party Generative AI Model providers when you acquired your license to use them (for example, their terms of service, license agreement, acceptable use policy, and privacy policy).

You are responsible for ensuring that your use of the Third-Party Generative AI Models comply with the terms governing them, and any laws, rules, regulations, policies, or standards that apply to you.

You are also responsible for making your own independent assessment of the Third-Party Generative AI Models that you use, including their outputs and how Third-Party Generative AI Model providers use any data that might be transmitted to them based on your deployment. AWS does not make any representations, warranties, or guarantees regarding the Third-Party Generative AI Models, which are "Third-Party Content" under your agreement with AWS. Generative AI Application Builder on AWS is offered to you as "AWS Content" under your agreement with AWS.